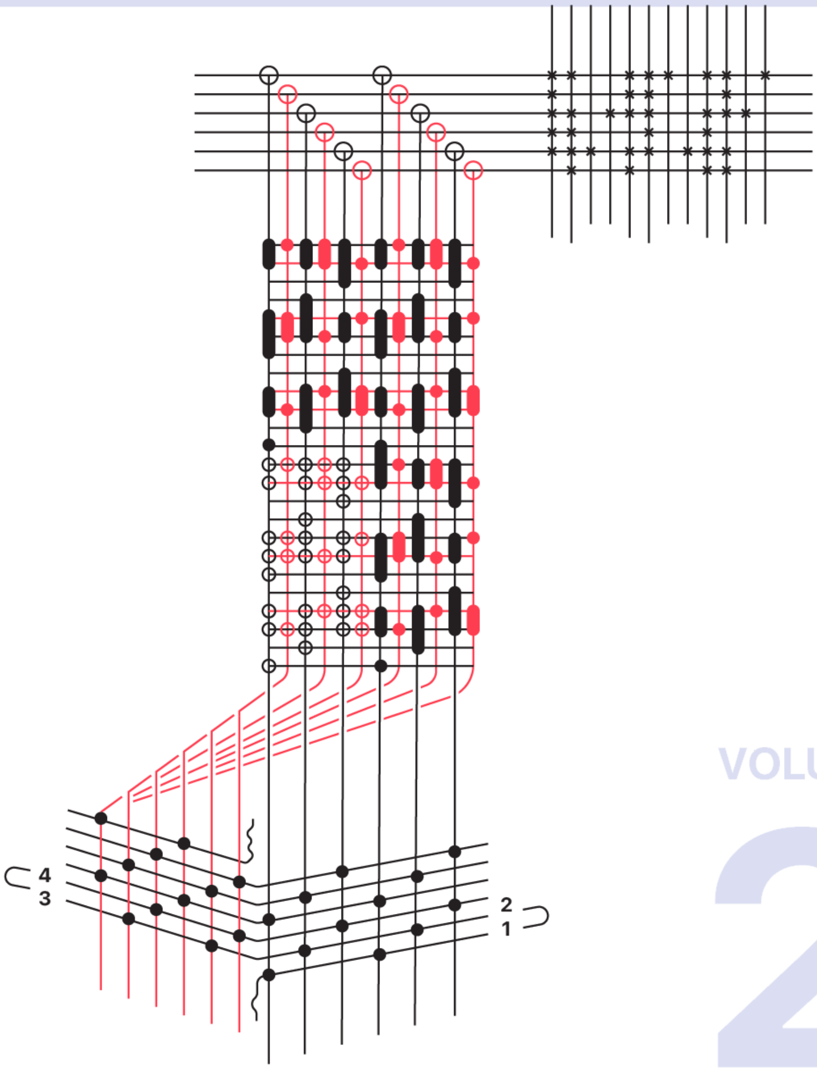
THREADS

The Evolution of Great Ideas in Computer Science



VOLUME 2

Faculty Coach
Sean Warnick

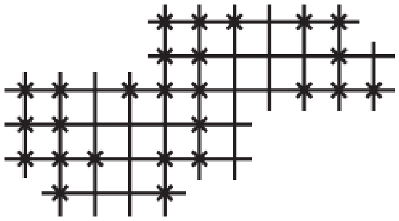
Editor
Michael DeBuse

BYU 2020

CS 611

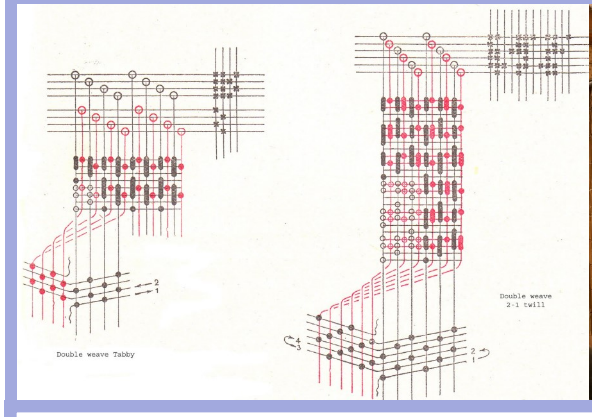
Welcome to

THREADS



The evolution of any great idea often begins long before its conception and implementation. Like thread woven from many individual fibers, people with varying backgrounds and education weave together the ideas that came before them with inspiration gained through individual investigation to develop the topics that inspire and guide our research today. THREADS investigates topics important to Brigham Young University's computer science graduate students or that inspire them to push the boundaries of discovery in their research.

Michael DeBuse
THREADS Editor



Front cover by: Gavin Jensen

Image repurposed from: https://historicweaving.com/wordpress/weaving-library/

Jacquard Loom, National Museum of Scotland

"The Jacquard mechanism, invented by Frenchman Joseph Marie Jacquard and first demonstrated in 1801, simplified the way in which complex textiles such as damask were woven. The mechanism involved the use of thousands of punch cards laced together. Each row of punched holes corresponded to a row of a textile pattern."

https://www.nms.ac.uk/explore-our-collections/stories/science-and-technology/jacquard-loom/

"The Jacquard loom utilized punch cards to aid in the automation of textile patterns. These punch cards were an inspiration for the first punch cards used to program early computer systems. The front image is a representation of the type of textile pattern that could be automated by the Jacquard loom. These beautiful patterns and textiles inspired the theme of this publication that weave together important ideas in the development of computer science."

Gavin Jensen

Contents

Syllabus 2020

The course CS 611 consists of three components: Tools, Papers, and Stories. The Tools are selections from Linear Algebra and Learning from Data by Gilbert Strang, taught by the Ph.D. candidates to each other, with coaching from the faculty advisor. The Papers are seminal papers in the field, as chosen by faculty in the department. The Stories section reviews the history of Computer Science as a discipline, mostly following The Universal Computer: The Road from Leibniz to Turing, by Martin Davis, and it culminates in a student exposition following the “thread”, or evolution of an idea of their choosing, published here. This section surveys details from each part of the course.

Tools

1. Highlights of Linear Algebra
    1. Multiplication Ax Using Columns of A
    2. Matrix-Matrix Multiplication AB
    3. The Four Fundamental Subspaces
    4. Elimination and $A = LU$
    5. Orthogonal Matrices and Subspaces
    6. Eigenvalues and Eigenvectors
    7. Symmetric Positive Definite Matrices
    8. Singular Values and Singular Vectors in the SVD
    9. Principal Components and the Best Low Rank Matrix
    10. Rayleigh Quotients and Generalized Eigenvalues
    11. Norms of Vectors and Functions and Matrices
    12. Factoring Matrices and Tensors: Positive and Sparse
2. Computations with Large Matrices
    1. Numerical Linear Algebra
    4. Randomized Linear Algebra
4. Special Matrices
    6. Graphs and Laplacians and Kirchoff’s Laws
    7. Clustering by Spectral Methods and k-Means
5. Probability and Statistics
    1. Mean, Variance, and Probability
    2. Probability Distributions
    3. Moments, Cumulants, and Inequalities of Statistics
    4. Covariance Matrices and Joint Probabilities
    5. Multivariate Gaussian and Weighted Least Squares
    6. Markov Chains
6. Optimization
    1. Minimum Problems: Convexity and Newton’s Method
    4. Gradient Descent Toward the Minimum
7. Learning from Data
    2. Convolutional Neural Nets
    3. Backpropagation and the Chain Rule

Papers

1. McCarthy, J., M. L. Minsky, and N. Rochester. "The Dartmouth Summer Research Project on Artificial Intelligence." *Dartmouth College, New Hampshire* (1956).
2. Kajiya, James T. "The rendering equation." *ACM SIGGRAPH computer graphics*. Vol. 20. No. 4. ACM, 1986.
3. Whitted, Turner. "An improved illumination model for shaded display." *ACM Siggraph 2005 Courses*. ACM, 2005.
4. Shannon, Claude E. "A symbolic analysis of relay and switching circuits." *Electrical Engineering* 57.12 (1938): 713-723.
5. Shannon, Claude Elwood. "A mathematical theory of communication." *Bell system technical journal* 27.3 (1948): 379-423.
6. Myers, Brad, et al. "Past, present, and future of user interface software tools." *ACM Transactions on Computer-Human Interaction (TOCHI)* 7.1 (2000): 3-28.
7. Ashish Vaswani, et.al. “Attention Is All You Need.” 2017
8. Nakamoto, Satoshi. “Bitcoin: A Peer-to-Peer Electronic Cash System. “ Cryptography 2009.
9. Samuel Grieggs, et.al. “Measuring Human Perception to Improve Handwritten Document Transcription.” 2020
10. Chomsky, Noam. "Three models for the description of language." *IRE Transactions on information theory* 2.3 (1956): 113-124.
11. L. G. Valiant. 1984. A theory of the learnable. Commun. ACM 27, 11 (Nov. 1984), 1134–1142.
12. Pearson, K. "On lines and planes of closest fit to systems of point in space." Philosophical Magazine 2.11 (1901): 559-572.
13. Needleman, Saul B., and Christian D. Wunsch. "A general method applicable to the search for similarities in the amino acid sequence of two proteins." Journal of molecular biology 48.3 (1970): 443-453.
14. Dreyfus, Stuart. "Richard Bellman on the birth of dynamic programming." Operations Research 50.1 (2002): 48-51.
15. Bellman, Richard. "The theory of dynamic programming." Bulletin of the American Mathematical Society 60.6 (1954): 503-515.
16. Dhillon, Inderjit S. "Co-clustering documents and words using bipartite spectral graph partitioning." Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2001.
17. Gödel, Kurt. On formally undecidable propositions of Principia Mathematica and related systems. Courier Corporation, 1992.
18. Lee, Daniel D., and H. Sebastian Seung. "Learning the parts of objects by non-negative matrix factorization." Nature 401.6755 (1999): 788.
19. Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean. “Efficient Estimation of Word Representations in Vector Space.” 2013
20. Turing, Alan Mathison. "Systems of logic based on ordinals." Proceedings of the London mathematical society 2.1 (1939): 161-228.
21. Rivest, Ronald L., Adi Shamir, and Leonard Adleman. "A method for obtaining digital signatures and public-key cryptosystems." Communications of the ACM 21.2 (1978): 120-126.
22. Dijkstra, Edsger W. "A note on two problems in connexion with graphs." Numerische mathematik 1.1 (1959): 269-271.
23. Kleinberg, Jon M. "An impossibility theorem for clustering." Advances in neural information processing systems. 2003.
24. Blei, David M., Andrew Y. Ng, and Michael I. Jordan. "Latent dirichlet allocation." Journal of machine Learning research 3.Jan (2003): 993-1022.
25. Istrail, Sorin, and Solomon Marcus. "Turing and von Neumann’s Brains and their Computers." 13th International Conference on Membrane Computing. 2012.
26. Wiener, Norbert. Cybernetics or Control and Communication in the Animal and the Machine. MIT press, 2019.
27. Dempster, Arthur P., Nan M. Laird, and Donald B. Rubin. "Maximum likelihood from incomplete data via the EM algorithm." Journal of the Royal Statistical Society: Series B (Methodological) 39.1 (1977): 1-22.
28. Kalman, Rudolf. "Discovery and invention: The newtonian revolution in systems technology." Journal of guidance, control, and dynamics 26.6 (2003): 833-837.
29. Kalman, Rudolf Emil. "Contributions to the theory of optimal control." Bol. soc. mat. mexicana 5.2 (1960): 102-119.
30. Kalman, Rudolph Emil. "A new approach to linear filtering and prediction problems." Journal of basic Engineering 82.1 (1960): 35-45.
31. Hastings, W. Keith. "Monte Carlo sampling methods using Markov chains and their applications." (1970): 97-109.
32. Rurnelhart, D. E., James L. McClelland, and PDP Research Group. "Parallel distributed processing.‘Explorations in the microstructure of cognition. Volume 1: Foundations." (1986).
33. Karmarkar, Narendra. "A new polynomial-time algorithm for linear programming." Proceedings of the sixteenth annual ACM symposium on Theory of computing. ACM, 1984.
34. LeCun, Yann, and Yoshua Bengio. "Convolutional networks for images, speech, and time series." The handbook of brain theory and neural networks 3361.10 (1995): 1995.
35. LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." nature 521.7553 (2015): 436-444.
36. Griffiths, Thomas L., and Mark Steyvers. "Finding scientific topics." Proceedings of the National academy of Sciences 101.suppl 1 (2004): 5228-5235.
37. Dantzig, George B. Origins of the simplex method. No. SOL-87-5. Stanford Univ CA Systems Optimization Lab, 1987.

Stories

MUHAMMAD IBN MUSA AL-KHWARIZMI 780 - 850
GALILEO GALILEI 1564 - 1642
JOHANNES KEPLER 1571 - 1630
CHRISTIAAN HUYGENS 1629 - 1695
GOTTFRIED LEIBNIZ 1646 - 1716
CHARLES BABBAGE 1791 - 1871
ISAAC NEWTON 1643 - 1727
$F_1 = F_2 = G\frac{m_1 \times m_2}{r^2}$
GEORGE BOOLE 1815 - 1864
The Ghost Club
Extractors Club
Analytical Society
PIERRE-SIMON LAPLACE 1749 - 1827
ADA LOVELACE 1815 - 1852
OLIVER HEAVISIDE 1850 - 1925
GEORG CANTOR 1845 - 1918
$\infty \ \aleph_0 \ \omega$
JAMES MAXWELL 1831 - 1879
1. $\nabla \cdot \mathbf{D} = \rho_V$
2. $\nabla \cdot \mathbf{B} = 0$
3. $\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}$
4. $\nabla \times \mathbf{H} = \frac{\partial \mathbf{D}}{\partial t} + \mathbf{J}$
GOTTLOB FREGE 1848 - 1925
$\forall ! \exists \rightarrow \neg \wedge \vee \Longrightarrow$
NORBERT WEINER 1894 - 1964
PARADOX
DAVID HILBERT 1862 - 1943
23 problems for a new century
BERTRAND RUSSELL 1872 - 1970
ABCDEFGHIJKLMNOPQRSTUVWXYZ
XAPZRDWIBMQEOFTYCGSHULJVKN
AMERICAN TELEPHONE & TELEGRAPH CO. AND ASSOCIATED COMPANIES BELL SYSTEM
HEAD
TAPE
VON NEUMANN 1903 - 1957
ALONZO CHURCH 1903 - 1995
ALAN TURING 1912 - 1954
KURT GÖDEL 1906 - 1978
recursively enumerable
context-sensitive
context-free
regular
CLAUDE SHANNON 1916 - 2001
RICHARD BELLMAN 1920 - 1984
WARREN MCCULLOCH 1898 – 1969
WALTER PITTS 1923 – 1969
RUDOLF KALMAN 1930 - 2016
IAS INSTITUTE FOR ADVANCED STUDY

Images for use in the collage were pulled from Wikipedia.com articles about the various people and topics covered in the course.

Computationalism versus Naturalism

Gavin Jensen
Brigham Young University

***Abstract*—Computational theories of mind are committed to the idea that the human mind is a computational system. This paper explores the history and philosophical assumptions behind computationalism and presents an important, though less well-known, objection to computationalism proposed by the philosopher John Searle.**

■ **ALTHOUGH COMPUTATION WAS INVENTED** to address specific problems in classical mathematics, it inspired new paradigms in philosophy, biology, and physics. These paradigms culminated in what can generally be called *computationalism*. Computationalism applied to the mind can be summarized by the oft-quoted phrase, "The mind is to the brain, as the program is to the hardware."[1] This *computational theory of mind* (CTM) assumes that the mind is a computer program. And the right sort of implemented program (natural or artificial) is sufficient to produce mental contents such as beliefs, thoughts, desires, intentions, and conscious experiences. John Searle named this view *Strong Artificial Intelligence*, but it also goes by the name *computer functionalism*. A closely related claim called *cognitivism* posits that the brain is a digital computer. In his 1990 essay, "Is the Brain a Digital Computer?" Searle presented a serious objection to computationalism. Computation, Searle argued, is *observer-relative* and therefore not eligible to count as an intrinsic feature of any physical system, including the brain.

These ideas are outlined in three main sections. Part 1 explores concepts of computation as inspired by Turing machines and information theory. Part 2 outlines a modern history of the philosophy of mind as it co-evolved with artificial intelligence, culminating in the computational theory of the mind. Part 3 outlines John Searle's primary argument against computationalism and situates that argument within the broader framework of scientific naturalism. This section also explores why Searle's objection is ultimately unanswerable by his naturalist critics. Finally, this paper will summarize important implications of these ideas for popular theories in biology and physics.

Part 1: Turing Machines, Information, and Computation

The birth of the computer, according to Martin Davis, begins with the philosopher and mathematician Wilhelm Gottlieb Leibniz, who invented calculus and the symbolic system of calculus used today. Leibniz dreamed of a universal characteristic that could calculate all truth by purely mechanical and symbolic means. This dream inspired philosophers and mathematicians for centuries and contributed to developments in symbolic logic.

In the early 1900s, the mathematician David Hilbert challenged the mathematical community to discover an algorithm that could prove whether any mathematical statement was true or false. In mathematics, a mathematical statement is called *decidable* if it can be proven true or false. If a solution could be found for Hilbert's 'decidable problem,' or *entscheidungsproblem* in German, then any mathematical question could, in principle, be answered by purely mechanical means. A

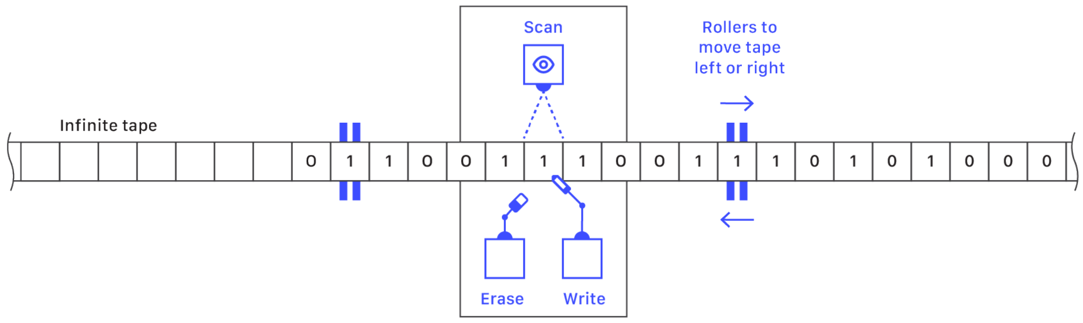


Figure 1: Components of a Turing machine

solution to the entscheidungsproblem would have, to a considerable extent, been "a fulfillment of Leibniz's dream."[2]

In 1936, the young mathematician and logician Alan Turing proved that "the Hilbertian Entscheidungsproblem can have no solution."[3] Turing's proof employed a novel thought experiment that imagined a hypothetical machine called an *a-machine* or automatic machine. Turing's proof says that (A) for every mathematical statement or problem that can be decided (proven true or false with an algorithm), that problem can be decided by an a-machine. Turing then proved that (B) there are some problems which cannot be decided with an a-machine. Therefore, (C) there is no algorithm that can decide every problem. Turing's a-machines later became known as *Turing machines* (TM).

The first part of Turing's proof states that for any algorithm, there is some TM that can implement that algorithm. This is called the Church-Turing thesis, independently developed by Turing and Alonzo Church. It was crucial in providing a more precise definition of an algorithm that could be used for mathematical analysis.

Turing Machines

Turing machines are abstract mathematical concepts that manipulate symbols printed on an infinitely long paper tape. The behavior of a Turing machine is "governed in part by internal mechanisms and in part by the specific symbols found on the tape."[4] TMs have four actions. They can read symbols, erase symbols, write symbols, and shift attention to new symbols by moving left or right along the tape. The state of the machine (its 'internal mechanism') dictates which action it will perform next. There are three states: a start state, an intermediate state, and an end or *halting* state that can output "accept" or "reject." Turing showed that any task could be recursively decomposed or broken down into ever smaller tasks. The smallest tasks were simple operations that could be performed using only two symbols, such as '0's and '1's. Turing imagined these symbols written down sequentially in separate boxes along the tape. A TM transitions from one state to another as it scans symbols on the tape and looks up which action it should perform next. For example, a TM could scan the symbol '1' on the tape, look up its current state, and then perform the action to erase the '1', write a '0', and move one square to the right. This process will continue, eventually reaching a halting state, or more interestingly, it will continue printing symbols forever without ever halting.

A universal Turing machine is a TM that can simulate any other possible TM. The concept of a Universal TM combined with the Church-Turing thesis entails that a Universal TM can compute any function that is computable by an algorithm.

Syntax, Symbols, and Semantics

Turing machines are called *syntactical* engines because they manipulate symbols. Syntax refers to the form of some representation such as symbols and sequences of symbols called strings. A TM uses algorithmic procedures to manipulate strings of symbols. According to Michael Sipser's Introduction to the Theory of Computation, "The input to a Turing machine is always a string. If we want to provide an object other than a string as input, we must first represent that object as a string. . . A Turing machine may be programmed to decode the representation so that it can be interpreted in the way that we intend."[5] A *description* of a TM is a string of symbols that defines that TM's function. These descriptions can be used as input for another TM.

Semantics is often contrasted with syntax. Semantics refers to the meaning that gets assigned to the symbols. You can have different syntax with the same semantics and vice versa. For instance, the English word "chair" has the same meaning as the Russian word, "stul" even though the syntax is different. Syntax has no intrinsic meaning or content. Consider the words you are reading right now. The symbols on the screen or printed page are just an arrangement of meaningless pixels or ink marks. They only have semantic meaning because you, as the reader, can do two things at once. First, you interpret the marks as symbols. For example, the ink marks or pixels that form the shape of a 'c' can be recognized as the letter *'c'*. Second, you interpret strings of symbols as having a specific semantic content or message. The string of symbols "c-h-a-i-r" can have the semantic meaning of referring to a particular "chair," for example. Algorithms act on the form, or syntactical structure, regardless of the semantic meaning of the forms.

Information

The history of computational theory goes hand-in-hand with information theory. In 1948, Claude Shannon wrote *A Mathematical Theory of Communication* which defined a binary digit or "bit" as a unit for measuring an exact quantity of information in some system. One bit of information can represent two possibilities. According to Shannon, "A device with two stable positions, such as a relay or a flip-flop circuit, can store one bit of information." In computer science, the symbols '0' and '1' are conventionally assigned to the physical states of a computational system. Technically, in a physical system such as a computer, there are no 0s and 1s. There are only certain voltages of transistors, for instance, that can count *as if* they were a '0' or a '1'. Applying these concepts to Turing machines, Shannon showed how to construct a universal Turing Machine with only two states.[2]

When Shannon presented his theory of information, he was not referring to information in the common usage of the term. In ordinary usage, information is constituted by some meaningful message or semantic content. Shannon with his collaborator Warren Weaver wrote that, "The word information, in this theory, is used in a special sense that must not be confused with its ordinary usage. In particular, information must not be confused with meaning," and that "the semantic aspects of communication are irrelevant to the engineering problem."[6]

Information in the semantic sense can be intrinsic or derived. Intrinsic information is information that conscious beings have. It is constituted by intrinsic mental processes such as perception and memory. Derived information is information that can be interpreted as having semantic meaning. The semantic information contained in the words you are reading now is derived. Information in the syntactic sense refers to the symbolic state transitions of Turing machines that are instantiated in a physical system. The term *information processing* is often used interchangeably with computation.[1]

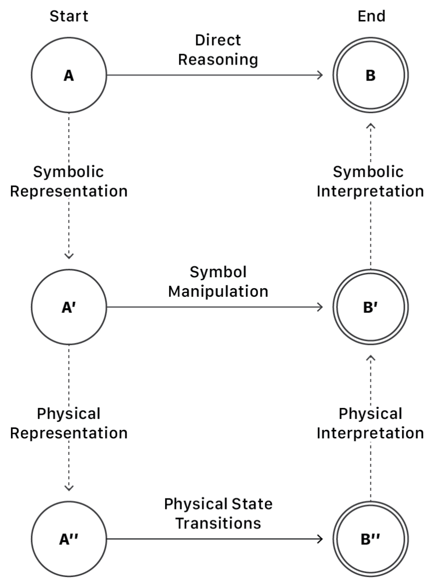

$$\begin{tikzcd}
A \arrow[r, "\text{Direct Reasoning}"] \arrow[d, dashed, "\text{Symbolic Representation}"] & B \\
A' \arrow[r, "\text{Symbol Manipulation}"] \arrow[d, dashed, "\text{Physical Representation}"] & B' \arrow[u, dashed, "\text{Symbolic Interpretation}"] \\
A'' \arrow[r, "\text{Physical State Transitions}"] & B'' \arrow[u, dashed, "\text{Physical Interpretation}"]
\end{tikzcd}$$

Figure 2: Levels of computation from implementation(bottom) to semantics(top)

[1]Technically, computation is only one type of information processing. Information processing can refer to effective computation, distributed processing, and dynamical systems processing.[7]

Direct Versus Symbolic Reasoning

Leibniz dreamed of a *universal characteristic* that could solve any problem and discover new knowledge by manipulating symbols using mechanical means. What is so important about using symbols? Why not reason about concepts directly? A partial answer suggests that reasoning can be simplified by using symbols. Symbols enable us to abstract away the meaning of concepts behind the symbols to focus on the logical relationships between the symbols. When the logical relationships are clear, we can more precisely and efficiently reach conclusions, expose faulty reasoning, and discover new knowledge. Understanding the relationship between direct reasoning and reasoning through symbolic processes illuminates different conceptions of computation.

Reasoning about Concepts

Reasoning about concepts directly involves seeing how a conclusion follows from a set of premises. For example, when presented with the premises, "Socrates is a man" and "all men are mortal," we can reason towards the conclusion that "Socrates must be mortal." The top level in figure 2 represents direct reasoning from starting premises ***A*** towards some conclusion in end state ***B***. This process of going from ***A*** to ***B*** can be generalized to include cognitive processes more generally such as perception, memory-retrieval, and belief-formation.

Intentionality

These cognitive processes are *semantic* because they are 'about' or 'directed at' something outside of themselves. We understand what the words 'Socrates,' 'man,' and 'mortal' refer to in reality. This ability of the mind to be 'about' or 'directed at' some object or state of affairs in reality is called *intentionality*. Intentional states include beliefs, desires, and intentions. A person's conscious mental states are intentionalistic because they involve an awareness of things that exist independently of the person. Thinking is likewise intentionalistic as it involves thinking 'about' something. A belief about Socrates is intentional because it is directed at Socrates, for instance. Often the word 'intentionality' is confused with intentions or goal-oriented activity, but intentions are only one type of intentionality.

Reasoning about Symbols

Reasoning about symbols enables us to arrive at conclusions without worrying about what the symbols are 'about.' We can represent the premises and conclusion of our example as follows:

Socrates is a man	X is a Y
All men are mortal	All Y are Z
Socrates is mortal	X is Z

In this case the premise ***A*** is represented as the string of symbols $\boldsymbol{A'}$. Applying simple rules to this string of symbols leads to a symbolic conclusion $\boldsymbol{B'}$. This symbolic conclusion can then be interpreted as having the semantic conclusion ***B*** that was intended. This process is shown in the middle layer of the diagram in figure 2.

There can be multiple levels of symbolic representation. The "**X**" in "X is a Y," for example, can be represented as the decimal representation "088"[2], and that representation could be represented in binary code as the string "01011000." Instead of having discrete symbols represent a specific semantic concept, these symbols could be represented in a neural network where the "information" constituted by a symbol is distributed across the weights and activations of the network. The syntactical level is all about manipulating symbols through algorithms. In mathematics, this process of linking semantics to syntax is called formalization. Proof theory shows how semantic relationships between propositions can be mirrored by syntactical relationships within certain well-known limits.[4]

Physical Implementation

The process of manipulating symbols to arrive at symbolic conclusions can be automated. This is possible through engineering physical systems that mirror the steps of the algorithms used to manipulate those symbols. A physical system can automate symbolic algorithms if (1) there is a starting and ending states $\boldsymbol{A''}$ and $\boldsymbol{B''}$ that map to our symbolic states $\boldsymbol{A'}$ and $\boldsymbol{B'}$ and (2) the physical state transformations map to our algorithmic procedures. The physical systems used to automate syntactic algorithms are typically called

[2]X is "088" according to the American Standard Code for Information Interchange or ASCII

computers. According to biologist John Mayfield, "A computer can be seen as a device in which one state (the input) interacts with another state (the current machine configuration) to produce a final state (the output)."[8] However, it may be clearer to talk about "computational systems" instead of "computers" to avoid being misunderstood as referring to the consumer devices found at Best Buy.

So far, the picture outlined above shows how semantic concepts can be represented as symbols. These symbols can then be mapped onto the states of physical systems. The benefit of engineering systems in this way is that they can automate some of our cognitive processes.

Multiple Realizability

An important aspect of computational systems is that the material composition of its implementation is irrelevant. Computational systems can be *realized* in (i.e., implemented in) any physical system or hardware that is sufficiently complex enough to carry out the steps of some algorithm. Computations can be implemented using electrical relays with binary states representing '1's and '0's. They can be made with marbles falling down tracks with gates that switch directions as marbles fall through.[3] Even a person consciously carrying out the steps of a Turing machine in their head or with pencil and paper counts as a computational system. More bizarrely, the philosopher Zenon Pylyshyn wrote that computational sequences could be realized in "a group of pigeons trained to peck as a Turing machine."[9] As long as the physical states correspond to (i.e., are isomorphic to) the states of a given algorithm, these examples are literally computationally equivalent. The principle that the hardware implementation is irrelevant to a computational system is the principle of *multiple realizability*—a concept explored more in part 2 and 3.

So What is Computation?

Computation can be defined in several ways. For the purposes of this paper, I will contrast what I will call *intentionalistic computation* with *Turing computation*. Intentionalistic computation is just another way to talk about direct reasoning. When someone consciously carries out the steps of some computation, they are performing intentionalistic computation. Mental math is a simple example. The original definition of "computer" was not an engineering artifact. 'Computer' was a job description for humans that perform computations, like an accountant or auditor. Intentionalistic computation may involve symbols, but the agent performing computation brings the interpretation to those symbols.

Turing computation is digital computation. Turing computation refers to the manipulation of symbols according to a set of rules where the symbols or bits of information are instantiated in some physical system. A physical system is said to be performing computation when it moves from one state to another according to the rules of some algorithm. The computational theory of mind assumes that intentionalistic computation is caused by Turing computations in the brain.

Part 2: From Dualism to Computationalism

The philosophy of mind aims to develop a logical foundation for understanding what the mind is and how it works. Trying to solve these problems has been described as "a veritable tangle of tangles."[10] Theories of the mind cannot be separated from metaphysical commitments about the nature of reality in general. Popular theories in the philosophy of mind evolved alongside scientific and technological innovations. The following account of the philosophy of mind outlines a rough historical background for computationalism, beginning with the philosopher and mathematician René Descartes.

From Dualism to Materialism

For Descartes, the primary feature of the human mind was the ability to think as expressed in his famous phrase, "I think, therefore I am." Descartes reasoned that the mind must be a separate substance from the body since "the body is by its nature always divisible, while the mind is utterly indivisible." This theory of mind, called "Cartesian Dualism" or "Substance Dualism,"

[3]You can build one of these yourself by purchasing the highly recommended learning toy *Turing Tumble*

posits two fundamentally different aspects of reality—mind and matter. Dualism, as described by philosopher Gilbert Ryle, is “the dogma of the Ghost in the Machine.”[11] One weakness of dualism is the lack of satisfying accounts for how the mind, which is immaterial, is supposed to interact with the body, which is material. This classic *mind/body problem* led many philosophers to adopt a naturalistic stance broadly called materialism. According to materialism, the mind, or whatever is constitutive of a mind, is not a separate substance but is part of the material world.

Behaviorism & the Birth of AI

An early form of materialism was “behaviorism”—the idea that the mind is constituted by behavior. Behaviorism posits that mental states are just dispositions towards certain behavioral outputs given certain environmental inputs. This theory is related to the “verifiability theory,” advanced by the logical positivists in the first half of the 20th century. The verification theory posits that the meaning of any statement just is its means of verifying that statement. Without some means of verification, a statement is meaningless. According to philosopher Edward Feser, “This theory seemed to make behaviorism almost unavoidable: if the only evidence you could have for verifying claims about what other people are thinking is the behavior they exhibit, then to say that they are thinking must be nothing more than to say that they tend to exhibit certain behaviors.”[8]

In 1950, when behaviorism was the dominant theory of mind, Alan Turing wrote “Computing Machinery and Intelligence.” In this paper Turing argued that a computer could be said to ‘think’ if it passes the imitation game. This imitation game, later called the “Turing test,” is a test of intelligence for computers. If a computer running a program can fool someone into thinking it is human, then it passes the test. Turing echoed the verification theory when he wrote, “The question, ‘Can machines think?’ I believe to be too meaningless to deserve discussion.”[12] This sentence is verificationist because it implies that the question of whether a machine can “think” can only be meaningfully framed in terms of some method for verifying that statement. The Turing test was designed to serve this role. Passing the Turing test does not just provide evidence that machines could think; passing the Turing test verifies the existence of the behavior that *constitutes* thinking itself.

Inspired by Turing’s vision, several researchers turned their attention toward building thinking machines. The term *artificial intelligence* was coined by John McCarthy, who wrote the proposal for the Dartmouth Summer Research Project on Artificial Intelligence in 1956. For some, the goal of artificial intelligence was not to create literal thinking machines but to more modestly produce computer programs that would simulate or mimic some aspects of human thought in order to perform useful tasks. This is called the “engineering approach” to AI. Others were far more ambitious and believed that computers were sufficient to create actual minds. The philosopher John Haugeland wrote, “The fundamental goal of this research is not merely to mimic intelligence or produce some clever fake. Not at all. ‘AI’ wants only the genuine article: machines with minds, in the full and literal sense.”[13] This second approach has been called the “cognitive approach” to AI. The engineering and cognitive approaches have been called the “two souls of AI”[14] which both claim a common heritage in the Dartmouth workshop.

From Behaviorism to Functionalism

Eventually, verificationism fell out of favor along with behaviorism. Verificationism failed because it was self-refuting. If statements are only meaningful if they were verifiable, then the verification theory must be meaningless since it cannot verify itself. A fundamental problem with behaviorism was that it could not account for consciousness or intentionality. Behaviorism led to the absurd conclusion that if being in pain just means that you are disposed to act a certain way, then the only way you could verify that you were in pain would be to look in the mirror and examine if you have any pain-related behaviors such as wincing or screaming.

Other materialist theories were proposed to remedy the failings of behaviorism. One such theory, called *Identity Theory*, argued that the mind is not just correlated with brain processes, but that mental states such as beliefs, desires, visual sensations, etc. are literally identical to brain processes. But this theory too fell out of favor for implying "neuronal chauvinism."[11] If a mental state was identical to some brain state, then that mental state could not be realized in a system that was not made of neurons in the exact same way. In other words, identity theories excluded *multiple realizability* for the mind.

In the 1960s, *Functionalist* theories were proposed to replace behaviorism and identity theories. Functionalism has been the dominant theory of mind since the 1970s. Functionalism embraces multiple realizability by modeling the mind, not in terms of what it is made of, but in terms of its functional and causal properties. It does not matter what a chair is made of as long as it can function as furniture for sitting. It is the same with mental states. The mind stands in a functional or causal relationship between a person's external stimuli (inputs) and behavioral outputs. Mental states such as perceptions, beliefs, desires, intentions, and sensation are functional states that can be realized in brain states.

The Computational Theory of Mind (CTM)

As early as 1943, Warren McCulloch and Walter Pitts developed a mathematical model of neural networks and suggested that the mind could be modeled by Turing machines. Since then, there have been several variations of the Computational Theory of Mind (CTM). In 1967, the philosopher Hilary Putnam argued for a form of functionalism called "Computer Functionalism" which argued that any creature with a mind is a Universal Turing Machine with probabilistic transitions between the states of the system. Computer functionalism rose in popularity along with the growth of artificial intelligence in computer science.

During the 1960s, psychologists, philosophers, linguists, and computer scientists formed a research agenda called "cognitive science." The tri-level hypothesis in cognitive science says that the mind can be analyzed with three levels. First, there is the implementation level constituted by the physical structure and organization in which mental processes are realized. Second, there is the algorithmic or syntactic level, where information processing (in the syntactic sense) was done. The third and highest level is the computational or semantic level of analysis which "asks what information processing problem (such as visual perception or higher-level thought) is being solved by a cognitive agent?" Cognitive science is "the study of 'mind as machine'", according to Margaret Boden, for "the core assumption is that the same type of scientific theory applies to minds and mind-like artefacts. More precisely, cognitive science is the interdisciplinary study of mind, informed by theoretical concepts drawn from computer science and control theory."[15]

CTM makes two related but separate claims. The first claim is that the human brain is a digital computer. Searle calls this claim *Cognitivism*. The second claim is that the mind is constituted by Turing computation. Zenon Pylyshyn wrote the "natural domain of human functioning (roughly what we intuitively associate with perceiving, reasoning, and acting)...can be addressed exclusively in terms of a formal symbolic or algorithmic vocabulary."[16] Searle named this claim *Strong Artificial Intelligence* because it makes the strong claim that "implemented software by itself, regardless of the nature of the implementing medium, is sufficient to guarantee the presence of mental contents."[9] This assumption is central to the cognitive approach to AI. Weak AI, by contrast, makes the weaker claim that computers can simulate or mimic human behavior without actually duplicating the semantic properties of human mental activity. Engineering approaches assume Weak AI.

CTM is generally treated not as a philosophical or logical issue but as an empirical question. It is typically presented as a materialist theory of mind, and it has had a powerful influence on the philosophical and scientific community. Jerry Fodor described CTM as "the only game in town." According to Searle, "to deny that the brain is computational is to risk losing your membership in the scientific community."[9]

Part 3: Computationalism Versus Naturalism

Argument Against Strong AI

One of the most famous thought experiments relating to Artificial Intelligence is

John Searle's Chinese Room thought experiment. Searle published the thought experiment in response to claims in the AI community that the programs were sufficient for genuine understanding. He imagined himself sitting inside a room, manually implementing the rules of a Chinese language program. As questions in Chinese were passed into the room, Searle would look at the Chinese symbols, consult a rule book, and then pass back symbols based on the rules in the book. To someone outside of the room, it appeared as though the room passes the Turing test. Searle, however, does not understand one word of Chinese. This thought experiment is intended to show that just shuffling symbols according to a set of rules is insufficient for understanding. The thought experiment was an illustration of a more fundamental logical argument called the argument against Strong AI. This argument, which can be stated independently of the Chinese room thought experiment, states that 1) based on the standard definition of Turing machines, programs are purely syntactical, 2) minds have mental contents (i.e., they have semantic content), and 3) "Syntax by itself is neither constitutive of nor sufficient for semantics." Therefore "programs are neither constitutive of nor sufficient for minds."

Premise 3 is the key premise. It states that since programs are defined syntactically (manipulating meaningless symbols), then there is no way to get from the level of syntax to the level of semantics. No matter how complex the symbol manipulation algorithm is, symbols cannot intrinsically be 'about' anything. They can only *derive* their meaning from the programmer or user of the computer. Searle is making a logical claim that no matter how much computers progress in the future, programs are logically not the types of things that could constitute minds since they lack any intrinsic semantics.

Two important claims come out of this analysis. The first is that semantics is not syntax. The second is that simulation is not duplication. Just as a computer simulation of a rainstorm will not get anyone wet, a mere simulation of thinking is not actual thinking.

Most responses to Searle attack the Chinese Room thought experiment and fail to address the logical argument. Searle's conclusion, however, is at best inconclusive because it can only addresses certain versions of CTM, while other variations remain immune to Searle's objection.[10] Exploring these variations of CTM is beyond the scope of this paper. Searle, however, has another less well-known objection to CTM that is more fundamental than the argument against Strong AI.

Naturalism

As mentioned in part 2, theories of mind are inseparable from the metaphysical commitments to the nature of reality in general. Searle's second objection to CTM is placed against the background of a prevailing paradigm that underlies the modern scientific world view. This general paradigm is called *naturalism*.

Science tends to proceed on the assumption of a naturalistic paradigm that assumes that we live in a mechanical universe where mindless, meaningless particles in fields of force interact with each other according to the laws of physics. To say that matter is meaningless is to say that it has no intrinsic or 'built in' meaning or purpose. Darwin's theory of natural selection—a cornerstone of naturalism—is widely interpreted as banishing teleology from nature. Green plants, for example, do not purposefully turn their leaves towards the sun in order to survive. According to natural selection, plants survive because they blindly happen to turn their leaves towards the sun. When we talk about plants or planets in purposeful vocabulary, that purpose is not part of nature. It is in the eye of the beholder.

Observer Independence vs. Observer-Relativity

There is an important naturalistic distinction between two classes of facts about reality. The first class of facts refers to that which exists independently of our thoughts, beliefs, and attitudes. Except for facts relating to the mind itself, if all conscious beings in the universe disappeared overnight these *observer-independent* facts would still exist such as the existence of atoms. Observer-independent facts are *intrinsic* or built-in features of reality.

The second class of facts refers to that which exists because we believe it to exist. This class of facts is called *observer-relative* facts.

Money is a classic example of observer-relativity. The existence of little green pieces of paper with chemical ink stains is observer-independent, but the fact that the paper counts as money is observer-relative. If everyone stopped believing that dollar bills are money, then they would cease to perform the function of money. Examples of these two classes of facts are included in table 1.

Observer-independent	**Observer-relative**
protons	politics
electrons	elections
molecules	money
mountains	marriage
tectonic plates	borders
photosynthesis	nation states
planets	religious rites

Table 1: Examples of observer-independent vs. observer-relative facts

So to which class of facts does computation belong? Does computation name some observer-independent process, or is it observer-relative? What about mental contents themselves?

Argument against Cognitivism

Physical science studies observer-independent features of reality. It is not concerned with meanings, purposes, or functions that people assign to those features. Social science typically deals with observer-relative phenomena, while psychology is somewhere in between. Intentionality is observer-independent. Beliefs, desires, intentions, and conscious experiences are intrinsic features of biological entities that have a causal structure sufficient for consciousness to emerge. Intentionalistic computation and intrinsic semantic information introduced in part 1 are intrinsic and therefore observer-independent. These mental processes exist intrinsically to some conscious agent independently of the attitudes and beliefs of anyone else.

But when it comes to Turing computation, could there be any observer-independent fact about a physical system that would make it computational? Consider the concept of syntax. What feature of a symbol makes it a symbol? Could something count as a symbol independently of what anyone else thinks or believes? No. Symbols are purely conventional. Something only counts as a symbol relative to our interests and conventions. To say that something can count as a symbol independently of our beliefs and desires implies an account of the universe that is more magical than naturalistic.

If symbols cannot logically be an intrinsic feature of reality, then Turing computation, which is defined by symbol processing, cannot be an intrinsic feature of reality either. The same goes for information in the syntactic sense. Of course, for any physical system that we count as a computational system, there are intrinsic physical states. The physical brain involves transfers of energy and computers involve transfers of energy. Aren't they similar enough to both cause mental states? A computational system made from transistors does have intrinsic properties such as different voltage levels in transistors and the production of heat. But as we have seen, the principle of multiple realizability entails that these features are completely *incidental* to the computational system. The same Turing computation could be realized in a group of pecking pigeons. The brain has an intrinsic causal structure sufficient for the emergence of mental states. But systems that we engineer to carry out the steps of Turing computation could not logically have these same mental states unless they also shared the same causal properties. This argument is more fundamental than the argument against Strong AI because it gets at the heart of the nature of symbols.

Observer-independent	**Observer-relative**
intentionality	symbols & syntax
intrinsic semantics	derived semantics
intrinsic information	syntactic information
intentionalistic computation	Turing computation

Table 2: Observer-independence of computational concepts

Someone might object and say that computation is not really about the syntax but about the state transitions of a physical system. This is a deviation from Turing's original definition of computation, but the redefinition, however, does not succeed. Searle writes,

> "On this view we don't really need 0s and 1s; they are just convenient

shorthand. But...this move is no help. A physical state of a system is a computational state only relative to the assignment to that state of some computational role, function or interpretation. The same problem arises without 0s and 1s because notions such as computation, algorithm, and program do not name intrinsic physical features of systems. Computational states are not discovered within the physics, they are assigned to the physics."[9]

Because Turing computation is observer-relative, any physical system is not just multiply realizable, but universally realizable. Universal realizability means that for any program whatsoever, there is some object with sufficient complexity that admits of some description under which it is carrying out the steps of that program. Thus, according to Searle,

> "the wall behind me is right now implementing the Wordstar program, because there is some pattern of molecule movements which is isomorphic to the formal structure of Wordstar. But if the wall is implementing Wordstar then if it is a big enough wall it is implementing any program, including any program implemented in the brain."

Therefore everything in the universe is a computational system. Nothing exists that is not also a computational system. This theory is called universal pancomputationalism.

So is the brain a digital computer? In one sense, the brain is trivially a computer since anything complex enough can be given a computational interpretation. But in the observer-independent sense, the brain is not a computer since nothing is intrinsically a computer except for conscious minds engaged in intentionalistic computation. Again, Searle is not making an empirical scientific claim. He is making a logical claim:

> "The point is not that the claim 'The brain is a digital computer' is false. Rather it does not get up to the level of falsehood. It does not have a clear sense. You will have misunderstood my account if you think that I am arguing that it is simply false that the brain is a digital computer. The question 'Is the brain a digital computer?' is as ill-defined as the questions 'Is it an abacus?', 'Is it a book?' 'Is it a set of symbols?' 'Is it a set of mathematical formulae?'"[9]

Searle is often misunderstood as being unfairly anthropocentric or biology-centric. To clarify, Searle is not saying that it is logically impossible to build an artificial mind. If it is possible to build a machine with intrinsic mental states, then those features would be observer-independent. They would be caused by the physical structure of the system. The problem with the computational approach is not that computers are too machine-like to cause minds; the problem is that they are not enough of a machine. An artificially intelligent machine must have the causal powers sufficient to guarantee conscious mental states. Syntax is not sufficient to guarantee mental states because syntax has no causal powers at all. This is because syntax does not name a class of objects that exist independently of our beliefs and attitudes.

Responses to Searle

One objection to Searle's analysis is that more is required for something to count as a computational system. It is not sufficient that there are just physical states that can count as the symbols "1", "+", "2", "=" with a corresponding state that could count as "3". What matters is that there is an isomorphism between the structure of a program as a whole and the causal structure of the entire physical system. This is not a serious objection to Searle because he explicitly acknowledges that implementing a given program requires that the implementing medium must be sufficiently complex to carry out the steps of the program. The fact that a sufficient causal structure is required does not undermine the claim that computation is observer-relative.

John Haugeland argues that Searle's claim that syntactical features of a physical system are observer-relative is falsified since there are empirical tests for whether some system has those features.[17] This also misses the point because there can be empirical tests for phenemena that are observer-relative. You could imagine a rigorous empirical test for whether or not an object was a chair, for example, but whether anything could count as a chair is still just a matter of convention, and thus observer-relative.

Implications

Searle's objection to CTM exposes serious issues for the cognitive approach to AI in computer science. Trying to create truly intelligent agents by programming computers is off to a flawed start. According to Searle, AI researchers need to first understand how the brain causes consciousness and intentionality. Then they would need to engineer a machine that would duplicate those causal processes. Typing into a terminal cannot create artificial intelligence in the literal sense. Computer science can help us create clever fakes that may even become so advanced that they could pass the Turing test. Such systems, however, would be zombies; they may behave intelligently but without intentionality. Some thinkers, such as the futurist Ray Kurzweil, propose immortality through computers. If the CTM were correct, then we can just upload our minds to computers so we will live forever in a virtual reality utopia. If Searle's analysis of computation is correct, then these projects cannot succeed.

Although Searle's argument is directed at computationalism as it relates to the mind, his argument is just as applicable to computationalism in biology and physics. The famous evolutionary biologist Richard Dawkins gave an eloquent example of computationalism applied to biology. "What lies at the heart of every living thing is not a fire, not warm breath, not a 'spark of life'. . . It is information, words, instructions. . . if you want to understand life, don't think about vibrant, throbbing gels and oozes, think about information technology."[18] In physics, John Archibald Wheeler, who coined the phrase "it from bit," argued that information was primary giving rise to "every it—every particle, every field of force, even the spacetime continuum itself" and that physics will ultimately come to understand that "all of physics is the language of information." According to MIT professor Seth Lloyd, "Every physical system registers information, and just by evolving in time, by doing its thing, it changes that information, transforms that information, or, if you like, processes that information." Summarizing these ideas, Gleick wrote, "The laws of physics are the algorithms. Every burning star, every silent nebula, every particle leaving its ghostly trace in a cloud chamber is an information processor. The universe computes its own destiny."[18]

If we are to take these claims in biology and physics about information and computation literally as opposed to metaphorically, then Searle's argument presents significant philosophical issues. Computational *models* in psychology, biology or physics can be extremely useful in understanding the phenomena studied in those fields. Within a naturalist scientific paradigm, however, conflating natural processes with Turing computation is to confuse the reality with the model. For most problems in computer science, these problems are not an issue. Computationalism is irrelevant to the project of building a self-driving car, for instance. These issues are only relevant if people believe that computation has some psychological or metaphysical significance.

Conclusion

The story of computationalism begins with Leibniz's dream that eventually led to Turing's original thought experiment. Turing machines have inspired many important theories in computer science and the philosophy of mind. These ideas, combined with information theory, were adopted in biology and physics. Searle's critique shows that 'computation' and 'information' are not intrinsic features of reality as these concepts are fundamentally observer-relative. This objection ultimately rests on a naturalistic paradigm that is widely assumed in modern science. If naturalism is correct, then Searle's critique of computationalism is ultimately unanswerable given how computation has traditionally been defined. This should at least motivate Searle's naturalistic detractors to more closely examine our default scientific paradigms and definitions of computation. Perhaps this tension will inspire new paradigms that go beyond naturalism and computationalism.

Acknowledgments

Very grateful for Michael Arts, Michael Hansen, Melanie Jensen, and Sean Warnick for

their encouragement, advice, and correction in the process of writing of this paper.

REFERENCES

1. P. Johnson-Laird, *The Computer and the Mind: An Introduction to Cognitive Science*. Harvard University Press, 1988. https://books.google.com/books?id=Tf5gRFgVuegC.
2. M. Davis, *The Universal Computer: The Road from Leibniz to Turing*. CRC Press, 2018. https://books.google.com/books?id=azXSBQAAQBAJ.
3. A. M. Turing, "On computable numbers, with an application to the entscheidungsproblem," *Proceedings of the London mathematical society*, vol. 2, no. 1, pp. 230–265, 1937.
4. S. Horst, *Symbols, Computation, and Intentionality*. Steven Horst, 2011. https://books.google.com/books?id=3Xi_8t0lqHoC.
5. M. Sipser, *Introduction to the Theory of Computation*. Cengage Learning, 2012. https://books.google.com/books?id=1aMKAAAAQBAJ.
6. C. Shannon and W. Weaver, *The Mathematical Theory of Communication*. University of Illinois Press, 1998. https://books.google.com/books?id=IZ77BwAAQBAJ.
7. L. Floridi, *The Philosophy of Information*. OUP Oxford, 2013. https://books.google.com/books?id=I8RoAgAAQBAJ.
8. E. Feser, "From aristotle to john searle and back again: Formal causes, teleology, and computation in nature," *Nova et vetera*, vol. 14, no. 2, pp. 459–494, 2016.
9. J. Searle, *Philosophy in a New Century: Selected Essays*. Cambridge University Press, 2008. https://books.google.com/books?id=EW521XwPg-EC.
10. E. Feser, *Philosophy of Mind: A Beginner's Guide*. Beginner's Guides, Oneworld Publications, 2006. https://books.google.com/books?id=RBq9DwAAQBAJ.
11. D. Chalmers, *Philosophy of Mind: Classical and Contemporary Readings*. Oxford University Press, 2002. https://books.google.com/books?id=rtEo8PRjpVoC.
12. A. M. Turing, "Computing machinery and intelligence," in *Parsing the turing test*, pp. 23–65, Springer, 2009.
13. J. Haugeland, *Artificial Intelligence: The Very Idea*. A Bradford book, MIT Press, 1989. https://books.google.com/books?id=zLFSPdIuqKsC.
14. L. Floridi, *The Fourth Revolution: How the Infosphere is Reshaping Human Reality*. OUP Oxford, 2014. https://books.google.com/books?id=hOF_AwAAQBAJ.
15. M. A. Boden, *Mind as Machine: A History of Cognitive Science*.
16. Z. W. Pylyshyn, "Computation and cognition: Issues in the foundations of cognitive science," *Behavioral and Brain Sciences*, vol. 3, no. 1, pp. 111–132, 1980.
17. J. Preston and M. Bishop, *Views into the Chinese Room: New Essays on Searle and Artificial Intelligence*. Clarendon Press, 2002. https://books.google.com/books?id=jfLuDwAAQBAJ.
18. J. Gleick, *The Information: A History, a Theory, a Flood*. Vintage Books, 2012. https://books.google.com/books?id=WAKWDwAAQBAJ.

Gavin Jensen is a master's student studying computer science. He currently works for Intuitive Surgical designing interfaces for doctors performing robotic surgery.

A Framework for Intelligence

N. D. Munson
Brigham Young University

***Abstract*—There are various measures of intelligence that exist, but no framework has been constructed to compare or unite these frameworks. In this paper, such general framework for understanding how to measure intelligence is proposed. The basic foundational concepts of Function, Choice, Structure, and Value Framework are discussed, along with some of their conceptual historical heritage.**

INTELLIGENCE has many definitions. One of my favorite quotes about it is that "[T]here seem to be almost as many definitions of intelligence as there [are] experts asked to define it" (from R. L. Gregory as cited in [1]). However, if we extend our view to ask the question "How might we measure if an action is an intelligent one?", we might be able to use the resulting insights to guide research toward the much more studied question of what conceptual mechanisms cause such intelligent actions to be taken.

By combining over 70 definitions of Intelligence from multiple resources including psychologists and AI researchers, Marcus Hutter and Shane Legg concluded intelligence to be "[a measure of] an agent's ability to achieve goals in a wide range of environments." [1]. They note that common features included in other definitions "such as the ability to learn and adapt, or to understand, are implicit in the above definition as these capacities enable an agent to succeed in a wide range of environments" [1]. They later formalized this definition mathematically [2], and subsequently it was expanded upon by Ben Goertzel in an effort to mathematically characterize real world general intelligence by generalizing some of the functions and techniques they used [3]. All of their efforts were characterized in the framework of Reinforcement Learning, with their Agents exposed to an environment wherein they could take actions and receive rewards.

In this paper, we will create a more generalized framework than they used, for the purpose of allowing for further research on understanding intelligence that can be generalized to more domains beyond Reinforcement Learning (like those of optimal control, neuroscience, traditional machine learning, or dynamics to name a few). This will encapsulate the Reinforcement Learning ideas that were studied by Hutter and Goertzel but will also apply to a more general set of ideas where rewards may not be explicitly defined, or the structure and functions available to agents and their environments are not fixed.

CONCEPTS

There are three major concepts which we will explore for the intent of understanding intelligence. These concepts are those of *Function*, *Choice*, and *Structure*.

We will explore these ideas by using three concrete running examples that will exemplify how these pieces fit together under various contexts, and throughout the text, other side examples will be used as well.

The first example we will explore is that of the Turing Machine, in which the set-up is a Turing Machine consisting of a head that can read, write and maintain state, a tape upon which the head can read and write symbols, and the key component of the set of quintuples represented by a Turing table that serve as "instructions" for the Turing Machine, as defined by Turing when he created the conceptual idea as a Masters student for the purpose of proving the uncomputability of the decision problem [4] .

The second example is a deep reinforcement learning (RL) agent seeking to play a set of Atari games (whose possible actions include moving a joystick left, right, up, and down, and pressing a button), and is implemented with an end to end neural network architecture. Such was first achieved in 2013 by a variant of a deep Q-learning algorithm [5], and the test of creating control agents to play Atari games has continued as a fruitful benchmark up till the present day.

The third example is that of a fictitious human named John Doe who is experiencing an ethical predicament. John is experiencing the trolley problem, where he finds himself in front of a train yard lever which controls the tracks upon which a train is heading toward three people who are tied up and unable to move, but if the lever were pulled, the train would change to a side track and hit a single individual who is standing on that side track [6].

The selection of these three examples was chosen so as to give breadth to the scenarios that can be covered by understanding the framework which is given in this paper. Of particular interest to note is that the Turing Machine example is in itself very broad, and can be assumed to represent many different things, such as might appear in standard computation, or in many machine learning scenarios.

For example, the Turing Machine may be programmed to solve a clustering or regression problem, where the initial state of the symbols on the tape represent particular data points to be computed on, and the Turing table instructions are such that a particular algorithm will be employed (such as single linkage clustering or logistic regression) and the respective “answers” (clusters or an equation) will be the result.

We will explore the concepts of function, choice, and structure using these three examples as practical motivation, and then we will explore how together these concepts will allow us to understand a more general framework for intelligence. For each concept, we will delve into the hierarchical nature of the ideas it represents.

FUNCTION

ACTIONS

First, we explore the concept of the Function of a component, element, or process, or generally speaking, any “thing”. The driving question that is answered by exploring this concept is “How?”, meaning “How does it happen?”. At its most “surface level”, the function of a “thing” is equivalent to what Actions it performs, or the effects it may exhibit due to an Action affecting it.

As a real world example, consider that the Action often completed with a toaster is to toast bread. Therefore, what makes toast come out of the toaster when bread is put in: how does that happen? By the Action of the toaster toasting the bread! The Action taken by the shoe of a runner is to take the impact of the foot on the ground and disperse it and soften it such that the runner feels more comfortable.

This pattern continues. We might say that Action is exemplified in the head of a Turing Machine reading or writing of a symbol. Action is taken when the RL Agent sends a signal to press the button causing the onscreen character to throw a punch. For John Doe, the processing and thinking through of possible outcomes of pulling a lever also is an Action.

Thus, an Action can be defined as a process of any finite duration that updates the system being observed (updating either the physical structure of the agent or the environment). Indeed, any change to the state of the system could be described as the occurrence of some Action, as would the lack of any change.

In the last example with John Doe, the Action may have been imperceptible to the outside world, but it is still an Action in our stipulative definition, as would be the Action of waiting for something to happen like the train to arrive, or waiting for a signal such as a phone call to occur. These Actions of waiting can be thought of as maintaining some (likely non-equilibrium) steady state dynamics within the agent. However, using the ”update function” of making no change, such as if our agent were a rock sitting on the ground, would also be an Action. If the rock were rolling down a hill, the Action it takes is that of following the path given it by the curvature of mound on which it rolls.

MECHANISMS

If we insist upon knowing a better explanation for how our toaster toasts bread, we may again ask the question “How does this happen?” When we ask “how”, we are looking to find the understanding of the process in question in terms of the Mechanisms that drive it. We want to see what is driving the states progression through time.

For instance, when asking how a particular line of code is executed in a modern computer program, we seek to understand the processes that cause the language to compile and execute upon the variables in memory at that time in the way it does.

Concerning how our bread becomes toast, we may come to the more adequate answer of “bread becomes toast through the chemical reactions of the amino acids and sugars in bread from the transfer of electric energy to thermal radiation”. There’s a lot that could be further unpacked there, and thus a single Mechanism may comprised of many sub-Mechanisms.

The Turing Machine reads, writes, and interprets symbols through information manipulation as dictated by the instructional set of quintuples (since a Turing Machine is a conceptual construct, the literal implementation of any of its Actions is intentionally undefined so as to be as abstract as possible; thus we look to the definitions of its processes to understand how the “physical process occurs”, thus leading us to the answer of information manipulation, perfectly executed without fail, and existing in the realm of abstract thought.)

Our RL agent makes a particular choice through execution of each of the computational functions that comprise it: if it is in the process of training, we may say that the updates to the weights occur through the application of the back-propagation algorithm [7]. The back-propagation algorithm was created in 1986 by Rumelhart, Hinton, and Williams for the purpose of adjusting the numerical values of the connections in a neural network to create new features for learning. Thus we are consistent with their previous interpretation of the weight adjustment process when we say that changing network weights forms the Mechanism whereby the agent ”learns”.

THE LAWS/ BASE TRUTH

In each of the examples we have given, we have peeled back the metaphorical onion one layer, looking for the physical (or abstract counterpart in the case of the Turing Machine) Mechanism that executes from an Action taken. When we state that there’s more to be unpacked in the toast example above, we might wonder exactly how much more could be unpacked. If those process explanations have further explanations, then those explanations might as well, so where is this line of explanations to end? If a child were to enjoy this line of learning (about the toaster for example) longer than a parent feels capable of answering, the answer eventually becomes “I don’t know, go ask your Mom/Dad”, or perhaps for the more subtle caregivers “Modern Science hasn’t figured that out yet”, or perhaps simply the word “Physics”. Truly, this last answer may be the most insightful of all.

In modern times we may have the most elaborate mathematical models of what drives physics, but these equations do not hold in themselves the “base truth” of the laws of nature. Even if they perfectly matched them (which they likely do not), these equations still only represent these laws, and do not constitute them in themselves. Indeed, for any scenario in which we are observing the physical world (as in the toaster example or in the example of John Doe) we will ultimately work down the layers to a point where the mechanical explanation of what occurs is that the laws of nature are such that this is what happens.

Ultimately, it is the laws or base truth physics that governs what Mechanisms are possible in our temporal, mortal experience. This is not to say there is no value in approximating these base rules to the best of our knowledge, but when we do so, we are inherently observing a different scenario than the exact one we are trying to model. This is because in those computational models, as in with the Turing Machine and with the Reinforcement Learning Agent, the Base Laws are as we define them to be (assuming they have been implemented correctly).

In the case that we were to consider the Turing Machine only in the mathematically constructed realm, it's Base Laws would simply be the mathematical framework in which it was defined, in which case the internal consistency of the axioms leading to this framework becomes of great importance. In the physical world, we need not worry about the consistency of the laws or base truth of physics quite as much, because through empirical evidence we can see they are consistent enough to at least locally maintain the conscious experience we enjoy.

CHOICE

ACTION SELECTION

We now move to a discussion on the concept of choice. The main interrogating question driving this section is "Why?", which is asked to understand the more global context in which a process lies. For example, when asking why a particular line of code is executed in a computer program, the answer we may seek is that it executes because it was called from the line before, or perhaps because it is part of a loop or function that is executing. From seeing the multiple levels of what "functions" can represent from the previous section, we may suppose that likewise the object this "why" question is directed to could also apply to many levels of depth.

If a tennis player loses gusto and slows down before reaching a returned ball bouncing just out of reach, we may ask her the reason for her lacking performance. "Why did you slow down?" Here, she may state "Because I decided to slow down". This simple response parallels the response she may give for "How did you slow down?" (which question comes from the previous section on Function) by with her responding "By slowing down". In this manner, the question "How" directs one to delve deeper into the layers of the onion of Mechanism, and the question "Why" directs one to broaden understanding from the higher onion layers of context.

The idea of selecting an Action may be in some situations very rudimentary: this selection may not be a "choice" at all in the colloquial sense if there is but one deterministic option. Consider the Turing Machine, which makes the choice to follow it's instructions because it is defined to do so (as discussed in the Mechanism section above), so we may say that the Turing Machine itself does not really have any "choice" or "control" at all. In other words, it is deterministic, and "selects" the only option available to it.

In the case of our Atari agent, the choice to press the button was selected from the space of all possible Actions that could have been chosen (left, right, up, down, or button).

For John Doe, if he were to choose to pull the lever, that would be selected from the space of all Actions that *he* could have taken. He could have also chosen to run away, close his eyes, call for help, or do anything else that would be possible given his circumstance and the Base Laws of physics.

SELECTION POLICIES

The question of "why" leads to an explanation of the context of a particular Action by illuminating other possible Actions. In this way, we can "step out" of the Action and see what other Actions would have been possible to choose.

When in the perspective of this "stepped out" view, there may be an infinitely large set of permissible Actions given the Base Laws. This does not mean that an agent is always able to assess or model all possible Actions. Indeed, the agent may even have an internal model about the Base Laws (or the Mechanisms founded on them) that is incorrect. (This incorrect model may lead the agent to select a particular Action and experience unexpected results.) Therefore, this "higher", "stepped out" view of the context an Action is performed under leads to an understanding of what else could have happened and forms the foundation of selection policies.

Return to the tennis player. Her "first level" response to the question of why she slowed down, "Because I decided to," can be understood to mean that she selected the Action of slowing down. This does not provide us with a "higher" view of context to see other Action options she could have chosen. However, she could also respond "Because I was getting tired". This second response provides a way to step one layer out again to the "second level". It describes the context from which her Action of "slowing down" was selected, and simultaneously provides a view into the other Actions that would be available in this context. Knowing that she was in the context or following the policy of being tired, we then know what other possible Actions she might have consequently taken. To think of a few, she could have instead collapsed, cried, laughed, or abruptly stopped running.

A quick explanation is necessary to describe what the term ”policy” means is as it was just used. In Reinforcement Learning, a policy is a mathematical strategy or function that an agent employs in order to select what Action to take. The idea of iterating policies as a way for learning was first introduced by Ronald Howard in 1960 as a way to solve Markov Decision Processes (MDPs)[8]. Though our situation is not one of an MDP, here similarly, a policy entails a function for selecting what Actions to choose, as well as the method for choosing them. Because their purpose is for selecting which Action to choose, we call them “Selection Policies”.

Each higher policy layer provides a wider “view” of why a particular chain of Mechanisms is used. We will look at a “third layer” of our Tennis player example to explain why she took the Action of “slowing down”. This third layer may reveal she was feeling tired because she had practiced tennis for many hours. If this were the case however, then she might have chosen to feel something other than tired: being tired was only one of the many Actions she could have chosen in that context. If she instead had chosen to feel determined, confident, or satisfied from her many hours of hard work, each which would have had in turn its own separate subsequent Mechanisms and Actions available.

In this way, each higher policy selection level represents a fixed perspective from which lower Actions and Mechanisms can be selected. Given that a particular policy is fixed, such as our tennis player “feeling tired”, the options of what Actions she could then take in response, and the method of her choosing them are fixed. The algorithm for selecting the Action may be sampling from a probability distribution over the options, or it may be a different algorithm (computationally permissible according to the Base Laws on which she operates). Either way, due to the fact we have fixed the policy “feeling tired”, then the corresponding algorithm of selection pertaining to that particular policy is also fixed. Each selection of an Action leads to a new policy to implement to select the Mechanisms by which it will operate, but not every selection path needs to have the same “depth” to reach the Base Law.

BASE CONTROL/FREE WILL

It should be clear that Selection Policies really are the same “kind of objects” as Mechanisms, only viewed differently, as will be explained below. Therefore, when we ask “Why?”, for the purpose of discovering a Selection Policy for a particular Action (or process) a, we are in a sense asking, “What Selection Policy perceives the Action a as a potential Mechanism?”. There may be many different Selection Policies that contain a in their set of possible Actions, and thus there is not a unique answer to this question. However, there *is* a unique policy (and linked chain of policies up the hierarchy) that an agent is employing each time that an Action is performed. Likewise, the Action a can be seen as the Selection Policy over the set of potential Mechanisms that would realize it. Thus, the concepts of Selection Policy and Mechanisms, or those of Function and Choice, are two descriptions for the same set of “explanations”. Viewed from the Function perspective, these “explanations” are understood as Mechanisms, and viewed from the Choice perspective, these “explanations” are understood as Selection Policies.

With this understanding, we can see that something is deterministic if it is bounded above in the “policy” direction of explanations, meaning that there is an explanation for which no higher description of “Why” can be given other than that the policy is fixed as such. On the other hand, theoretical constructs of formal abstraction are “created universes” that exist by bounding all explanations below in the “Mechanism” direction of formulation, meaning that there is a set of Mechanisms for which no lower description of “How” can be given, other than to say that the rules are fixed as such.

Our running example of the Turing Machine is an example of the latter, where the Mechanisms are bounded below. The rules and axioms have been defined for what the Turing Machine is and,

and how it operates, and as long as those rules serve as the base law for what the Turing Machine does, any number of explanations for “Why” our machine was implemented to act in a particular manner may be given. See Figure 1 for a visual on how this works.

As mentioned in the section on Action Selection, the Turing Machine follows a deterministic process. Therefore, the question of “Why did the Turing Machine make that choice” is trivial: ”The Turing Machine is deterministic, so it made that choice because that’s what the laws state is does in the context it is in.” A more meaningful answer may be given if we look not to the Turing Machine, but rather to the human creator of the Turing Table that controls it. They may be able to give a reason for why they made the choice to endow the Turing Machine with the initial conditions and set of instructions it was given. However, if we continue to press the question as to ”why” that reason needed to be filled, and continue further and further up the chain of ”whys”, our human must either come to some last policy, or continue forever higher up an infinite stack.

If we assume that the human in question is not deterministic, and thus is not driven by some fixed ”last policy”, we will arrive at the philosophical impasse of either requiring that there be “turtles all the way up”, or we arrive at a special kind of policy from which neither determinism nor a fixed method of sampling can computationally describe how the choice from that policy is made. (This “final Selection Policy” might be understood theologically as the soul, spirit, consciousness, or essence of the individual.) Either way, the empirical result of an infinite regress of Selection Policies or an ultimate “final Selection Policy” (which we call the Base Control) would functionally appear the same. The attribute that both the infinite stack and Base Control would display of not being either deterministic, nor even predictable, we define as “Free Will”.

As far as we can directly observe, it is the human coder who makes the decisions concerning why a particular Reinforcement Learning setup and policy is implemented, or why a particular update rule is employed to train the agent. “Meta-learning” may incorporate an extra layer

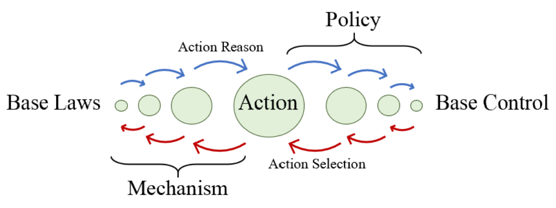


Figure 1. The Choice/Function Unification Diagram

of policy freedom, but it is still ultimately the Free Will of the coder being applied in timing of initializing the random seed that determines what the outcome of the training would be (unless it were stochastic, in which case, even those components are still determined not through free will but through deterministic mathematical functions) that would determine the final state of the trained agent.

However, this idea of meta learning has been used in practice, perhaps more notably in the paper on Agent57, where Deep Mind trained an Atari agent to use a meta controller to decide when to exploit known dynamics and when to explore. There, the RL agent was given two layers of policy control: the higher or outer layer so that it could control the use of its exploration/exploitation policies, thereby affecting the lower or inner layer to select which Action it should take. [9].

Of course, with the example of John Doe or with us, the question of whether we as humans are at the “base control” of our own Actions has been in philosophical dispute for centuries. What can be hopefully said with little disagreement is that we can observe our own behavior and determine that we perceive ourselves as having Free Will. Therefore, we may (reasonably, or egocentrically, depending on your perspective) declare our consciousness to either be the base source of control, or the highest rung on the ladder of policy driven control that we will account for, thus allowing any potential higher levels of control to be summed up in ourselves that we might have full claim over the phenotype of “Agents with Free Will”.

On the other hand, a Reinforcement Learning agent is of course programmed to have some fixed upper policy for updating it’s policies, as is determined by it’s hard-coded update method. However, the agent can also be evaluated “as if” it had Free Will in it's own right for practical purposes by treating the fixed policy "as if" it were a Base Control with Free Will.

STRUCTURE

OBJECT OF CONTROL

The next concept we will cover is that of Structure, with the purpose of distinguishing the structure and dynamics (or motion) of our agents in question in conjunction with and compared to their environments.

For the concept of structure, the driving question we ask is "What?", meaning "What is acting or being acted upon?" We can use the concepts of Function and Choice as discussed above to phrase this question slightly more technically: "What are the dynamics of the system resulting from the Base Control and Base Laws?"

Let us call the parts of the system the "components" of the system. There will be a subset of the components that comprise the agent, and the rest will comprise the agent's environment (which may include other agents with Free Will, but they are still part of the first agent's ”environment”). We define the Object of Control to be the components of the system that are directly controlled by the agent- they are likely a subset of the components that make up the Agent itself. It is through the Object of Control that the Base Control (or "spirit") of the Agent can interact with the Mechanisms and Functions that causally couple it to the indirectly controlled components that together with the Object of Control make up the entirety of the Agent.

An important point to emphasize here is that there is no conceptual boundary as to what inherently makes a particular subset of the components in a system an ”agent” verses what makes that set of components to be a part of an environment. This means that it is completely dependent on the frame of analysis as to which set of components (connected or not) can be viewed of as a ”single agent”. In this mindset, the environment can be thought of as being completely composed of agents, some of which are deterministic, and others which have Base Control. Such can be seen in the example given in the Actions section of this paper, where even a rock can be seen as taking the Action of updating its environment through its deterministic choice of not moving, or by following a path down a hill as determined by the Base Laws that we observe as gravity.

If we choose to focus on a particular agent (subset of the system) which does not have Free Will, then there is some highest level policy which is fixed and dictates what the dynamics of the Agent will be. These dynamics are then only dependent on the initial condition of the entire system, and the dynamics of the environment throughout time.

This is the case of the Turing Machine. Consider the agent to be the read/write head and Turing table, and consider the tape as the environment in which the agent is operating. In this situation, given the Turing table and tape, the Actions of the head are fixed; it's dynamics being completely dictated by the state it is in and its environment.

Indeed, even if we imagined that during the middle of the head's computation, we come in as an outside agent (with Free Will) and change some of the symbols on the tape, then given that we did this, the Turing Machine head's operations are still deterministically fixed assuming no other alterations to the tape were made.

These controlled dynamics can be very robust. For example, an agent might have "internal memory" if it can use it's Base Control to drive a function that can dynamically take information (external or internal to the agent) to compress or encode it, a function to maintain it over time, and another function to take that compressed or encoded information and utilize it in a meaningful manner (either by uncompressing it, or using its raw uncompressed information as part of another Function).

We might interpret the deep RL agent as doing this when it captures its experiences of various Atari games into memory, and uses these experiences to update it's policy for future action. An example of this principle in practice can be seen in Agent57 [9], as in many other RL agents.

A separate way for tracking memory may be external to the agent, not by changing its inner dynamics and Function, but rather creating an external memory in the environment, like a forgetful individual writing down notes with pencil and paper. These Actions are made easier because of the stagnant nature in our universe of objects that lack Free Will, due to the Base Laws that we experience. That is to say that it is easier to write with a paper and pencil due to the fact that Newtons first law of motion will keep papers and pencils from moving unless we are moving them.

In the case where the agent in question does have Free Will, it is necessary to note that the Base Laws must determine both the type of interaction between components of the system, and also what kind of interaction is feasible between the Base Control and the system as well. This means the possible ways the system might progress are dictated by the Base Laws and Base Control of our agent. If these Base Control/System interaction laws are in place as a part of the Base Laws, then our agent is free to act as he will, given the constraints of what dynamics are possible with the Base Laws.

HIERARCHICAL STRUCTURE

In the "Thousand Brains Theory of Intelligence", Jeff Hawkins and Christy Maver present a theory on how the human neocortex works, suggesting that the brain builds many models of every object. In the theory, each model comes from a cortical column of the human brain. (A cortical column is a neuroscientific construct that consists of a locally adjacent group of neurons in the human cortex.) This can be understood in our framework to mean that our brain is composed of many agents (each consisting of many neuronal components), each modeling the world we experience [10].

Karl Friston has done similar work in applying the theory of Markov Blankets to the brain in such a way that separates its dynamics in a hierarchical manner [11]. Indeed, he has done work that has shown that Markov blankets are useful in talking about the causal structure inherent in dynamical systems on multiple ”levels” of size, ranging from molecules to ecosystems [12]. (A Markov Blanket can be understood to be the set of components that separate any defined agent from their environment).

We build off these principles by further investigating the hierarchical nature of the structures that exist in our agents and the environment, and how they relate to the Functions defined above. In the case of John Doe, he is as you or I am: His body has a complex hierarchical structure (or Nested Structure) of organs, cells in the organs, organelles in the cells, and molecules and atoms comprising it all. However, he only has Free Will over the outermost grouping- the body.

This is not to say that the cells may not be agents with Free Will in the system as well, also with the capability of having some amount of control over the dynamics of what happen, the causal dependencies being separated by a Markov Blanket. The same applies for any layer of the hierarchical structure for that matter: the atoms, molecules, organs, or environment each may have free will or Base Control of their own, with the interactions of the way their dynamics driven by the Base Laws.

It may be possible to conditionally separate the system into a Hierarchical Structure many times over with Markov Blankets. At every level that this is possible, it is not necessary that the resulting collection of components really does have a Free Willed agent of it's own that acts on it through the Object of Control. For example, it is thus possible that John has an Object of Control, but that each of the separate molecules and proteins that help to build the structure that is his body do not.

We also note that the agent may take in components from the environment to itself, and may lose components to the environment. For example, we may eat broccoli and have those atoms become a part of what is on the inside of our Markov Blanket. Likewise we may cut our hair and clip our nails, and have them no longer be a part of what is on or inside our Markov Blanket.

AGENCY

The sociological concept of agency is the capacity for individuals to act with free will. We parallel this idea here and define Agency to be the described ability to utilize Base Control to direct the dynamics of the system via Actions chosen with Free Will.

From this, we would say that if we assume John has an Object of Control and Free Will, then because he has a body that is capable of following the dynamics the Functions and Base Laws allow it to, he thus has Agency.

If we ask whether the RL Atari agent has agency, again, it depends on how we chose to analyze it. If we treat it from the perspective that states that the highest policy is fixed, then we can quickly determine there is no Agency involved. However, if we treat it in the (more practical) framework “as if” the agent has Free Will (as was done in the section on Base Control/Free Will), then we discover that our RL agent does have an Object of Control. In this case, the Object of Control is any part of Neural Network that can be updated during training, as well as the controls that the agent uses to interact with the Atari games. It is only through these two sets of components that our “Agent” can update itself (while being trained) and interact with the universe (of Atari games). After the agent is trained, and the weights are fixed, the Object of Control then diminishes to be only the joystick and button controls that the agent has to interact with it’s world.

A thought experiment posed by Anatoly Dneprov in 1961 Russian Journal asks the philosophy of mind question what would happen if a large number of individuals were all asked to simulate the Actions of one neuron in a brain, were given the rules on how to do so, and the means to communicate in order to simulate the passing neuronal signals [13]. Would the resulting arrangement have a mind or consciousness (or Agency) in the same way we do?

Using the definition of Agency given above, the answer is no. This is due to the fact that each individual is making the “choice” to participate. Thus, there is no Agency in this scenario that has control over the dynamics of the group. Although the rules given to the participants may have given instructions on how to handle people deciding not to participate (which might be akin to a neuron in the brain no longer working), these instructions are assumed to be static, not dynamic, and therefore they represent an ultimate highest policy, or in other words, the simulation is deterministic (as described in the section on Base Control/Free Will).

FRAMEWORK FOR INTELLIGENCE

Now that we have covered the principles of Function, Choice, and Structure, we will investigate how these combine together under a larger framework of intelligence. It is key to note that here, while we will provide a method for measuring intelligence under any given Value Framework (as will be defined), we do not make an attempt to determine which Value Framework is the best.

DYNAMIC STATE PROGRESSION GRAPH

The first key to combine these principles together is that of Dynamic State Progression. This is the idea (that should seem familiar by this point) that given an initial condition for the system (agent and environment), and given the Base Laws that govern the dynamics of the system, there is a set of states that the initial system can transition into. For any of those states (where state means both structural organization of components and Mechanisms and policies being employed), there are another set of states that could be transitioned into (given again that the Base Laws are being followed).

Thus, the Dynamic State Progression Graph comprises all possible paths of the agent, given it’s initial starting conditions. (This principle is built off the idea of State Graphs presented by Steven Wolfram in his ”Physics Project” effort to find a fundamental theory of physics [14].)

If the Agent has Free Will and Base Control in influencing the dynamics of the Object of Control (in other words, if the agent has Agency), then the agent is the only thing that keeps the system as a whole from being deterministic unless the Base Laws have some stochastic elements to them or there are other agents with Free Will in the environment. If there are other Agents or the Base Laws themselves are not deterministic, then the Dynamic State Progression Graph is much bigger, but it still exists.

VALUE FRAMEWORK

Once we have the agent, and we have the system which comprises both the agent and the environment in which it resides, the framework we will define for this agent will provide a way to evaluate how intelligent the agent is. We will thus be constructing a Value Framework with respect to which we can measure the “intelligence” of the agent. This ”value” will only be defined for the ending state of the agent, but it follows much in the same way as standard dynamic programming problems are solved to optimize the final reward or minimize cost as presented by Richard Bellman in 1954[15].

Therefore, an agent’s intelligence as we define it here is dependent on what framework is being used to evaluate it. These Frameworks can be thought of as perspectives or points of view.

Given the fact that the system can progress to and toward different states depending on the choices made by the agent, there are many different terminal states and infinite horizons toward which the agent and its surroundings might progress. In selecting a Value Framework used for evaluating the agent given its initial surroundings and configuration, we do so by deciding the *relative preference of those terminal states and infinite horizons* (where infinite horizons refers to dynamic systems that is not finite in duration, and which the framework does not “make finite” in duration).

For example, suppose the agent is a robber and the environment is their world. If we choose to evaluate the robber from the Value Framework as might be seen from the robber’s perspective, the ideal end states might consist of successfully robbing the bank and escaping without getting caught, and the least desirable state would be that the robber gets injured, caught, or killed.

A Value Framework that might represent the perspective of society would likely have a different set of ideal final scenarios. From society’s perspective, the most desirable outcome would be one in which the attempted thief is quickly and safely recognized and captured, and the least desirable outcomes would include the thief getting away with lots of money and wrecking havoc.

It is important to note that once a framework is decided upon it remains fixed. So, if we evaluate the previous scenario from the perspective of the robber, we might do so by choosing a framework that represents the preferences of the robber before beginning to rob the bank. Using this framework, if partway through, the robber has a mid-robbing-life-crisis and decides it is a bad idea to rob the bank and instead decides to leave the bank without robbing it, the fact that his mentality has shifted midway through the process does not change the fact that his initial mindset (and thus the framework being used here) would have considered him leaving the bank without money a bad idea.

In a similar manner, we also note that because the Agent is completely separate from the Value Framework it is being evaluated upon, the relative ordering of all the potential terminal states and infinite horizons need not make any logical sense to the agent.

Thus, to evaluate how intelligent an agent is within a particular Framework, we simply note the value given to the terminal state or infinite Horizon at which the Agent arrives. That value represents the intelligence of the agent with respect to that framework.

If a system as a whole is deterministic, this simply means that the state of the system at the final time is completely determined by the state of the system at the initial time. Therefore, when placed in a Value Framework, the system can only follow one path, so the final state reached has its value completely defined by the Framework being used.

APPROXIMATION

Due to the large structure of these frameworks, we will rarely be able to compute to the whole thing in practice, so the best we can do is approximate. It may also be helpful to treat deterministic agents “as if” they had free will, as was done in the section on Base Control/Free Will. In practice with these “as if they had Agency Agents”, it may be necessary to test the agent multiple times on a particular framework and average the results.

Even if an agent with Free Will could be perfectly measured according to a Value Framework as described, it would not actually be indicative of how that agent would necessarily perform in the future, given the fact that an agent with Free Will need not follow the same policy every time, no matter how it has done in historical context.

Finally, we note that although an agent may ”perform well” with respect to one framework, this does not mean that the agent provided has intelligence in a general sense. To quote Ben Goertzel, ”intelligence is achieving complex goals in complex environments” [3]. Thus, we may create multiple ”Template Frameworks” with which to evaluate a single agent, by creating a set of initial conditions and terminal state rankings within which to place our agent. We can then use the combined results as a measure of the agents ”general intelligence”, although again, it is still with respect to the set of template frameworks that were employed.

CONCLUSION

There are many different factors at interplay that lead to intelligence. In this paper, we have identified four of them.

Function comprises any Action or interaction that is possible in the system given. Choice is understood as the reasoning or control that leads to the sequence of functions or rules that are applied to the agent. Structure defines the nature physical system or universe in which the Agent operates, and is closely coupled with the previous two. This is because Base Functions (or Base Laws) define what kind of interactions are possible in the system, and Choice, or the Base Control, determine the sequence of Actions implemented that results in the particular subset and ordering of interactions used. Finally, the concept of a Value Framework defines the perspective on which an agent or system is judged to be intelligent.

These four concepts provide an overarching framework with which one can understand intelligence in any system with temporal dynamics. They were selected to be exhaustive, meaning that there are no aspects of an agent within a system that cannot be framed in this context. However, this paper does not offer rigorous proof that these concepts are indeed exhaustive in this manner. Future work may include seeking to prove the completeness of these concepts, and implementing the principles of the overall Framework of Intelligence in a practical manner.

ACKNOWLEDGMENT

I would like to thank Jamison Moody for spending a good number of hours in discussion with me on many of these concepts, and helping me to flesh them out so they would make sense on paper. I would also like to thank my wife for listening to me even when my head is in the clouds and my feet are off the ground.

REFERENCES

1. S. Legg, M. Hutter, *et al.*, “A collection of definitions of intelligence,” 2007.
2. S. Legg and M. Hutter, “Universal intelligence: A definition of machine intelligence,” 2007.
3. B. Goertzel, “Toward a formal characterization of real-world general intelligence,” in *Proceedings of the 3d Conference on Artificial General Intelligence (2010)*, pp. 74–79, Atlantis Press, 2010/06.
4. A. M. Turing, “On Computable Numbers, with an Application to the Entscheidungsproblem,” *Proceedings of the London Mathematical Society*, vol. s2-42, pp. 230–265, 01 1937.
5. V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” 2013.
6. J. J. Thomson, “The trolley problem,” *Yale LJ*, vol. 94, p. 1395, 1984.
7. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *nature*, vol. 323, no. 6088, pp. 533–536, 1986.
8. R. A. Howard, “Dynamic programming and markov processes.,” 1960.
9. A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, D. Guo, and C. Blundell, “Agent57: Outperforming the atari human benchmark,” 2020.
10. J. Hawkins, M. Lewis, M. Klukas, S. Purdy, and S. Ahmad, “A framework for intelligence and cortical function based on grid cells in the neocortex,” *Frontiers in neural circuits*, vol. 12, p. 121, 2019.
11. K. J. Friston, E. D. Fagerholm, T. S. Zarghami, T. Parr, I. Hipólito, L. Magrou, and A. Razi, “Parcels and particles: Markov blankets in the brain,” *Network Neuroscience*, no. Just Accepted, pp. 1–76, 2007.
12. M. Kirchhoff, T. Parr, E. Palacios, K. Friston, and J. Kiverstein, “The markov blankets of life: autonomy, active inference and the free energy principle,” *Journal of The royal society interface*, vol. 15, no. 138, p. 20170792, 2018.
13. A. Dneprov, “Igra,” *The Game’), Znanie-sila (Knowledge is Power)*, vol. 5, pp. 39–42, 1961.
14. S. Wolfram, “A class of models with the potential to represent fundamental physics,” *Complex Systems*, vol. 29, p. 107–536, Jun 2020.

15. R. Bellman, “The theory of dynamic programming,” *Bull. Amer. Math. Soc.*, vol. 60, pp. 503–515, 11 1954.

Neal Munson is a Computer Science MS student interested in artificial intelligence. When he has leisure time, he likes to take a break.

Reinforcement Learning: An Approach to Intelligent Decision Making

Jamison Moody
Brigham Young University

***Abstract*—The ability to make effective decisions in everyday life is part of what makes us intelligent as humans. In the quest for artificial intelligence, this is one of the important skills we have tried to mimic. Reinforcement learning is an approach to decision making in computers that is loosely based on how reinforcing behaviors work in psychology. In this paper, we define reinforcement learning and explore the foundational ideas that started this paradigm. We also explore the equations, theorems, and common problems that are prevalent in the field of reinforcement learning.**

■ **HOW DO WE NAVIGATE** our environment to make effective decisions? The answer to this question becomes less and less clear as environments get more complex and complicated. This ability is important in every scenario of daily life. For example, when the alarm clock rings in the morning, we make the decision to get out of bed and get ready for the day. Later on, we may decide to stop whatever we are doing to grab a bite to eat. Do we stop at a new restaurant that looks appealing or do we go to one we know we like? Later on, do we decide to go to a movie with a group of friends or spend time on a work project we are putting off?

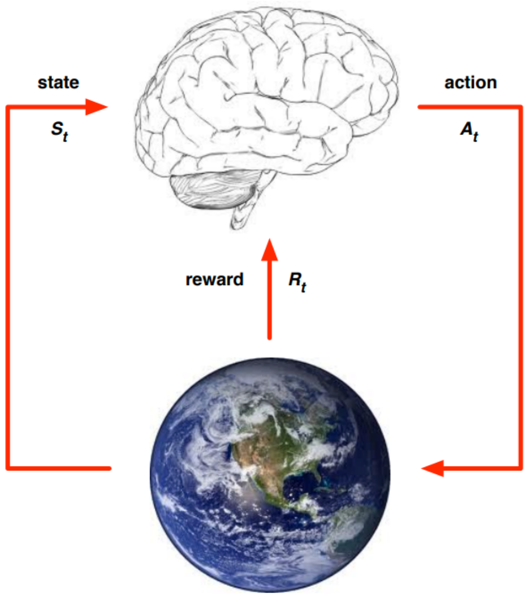


Figure 1. Reinforcement learning in a fully-observable environment. The agent takes an action based on the state of the environment. Behaviors are reinforced through rewards. Accessed from [1].

In order to make these decisions, we have to determine the value of the states we may end up in after we make the decisions. There is immediate reward (socializing with friends) or long term reward (finishing the work project). Is it worth it to try new food and find something new we really like or just eat what we already know we like (exploration vs. exploitation)? It is highly likely that our model of the world is biased, and not quite accurate. How do we update our

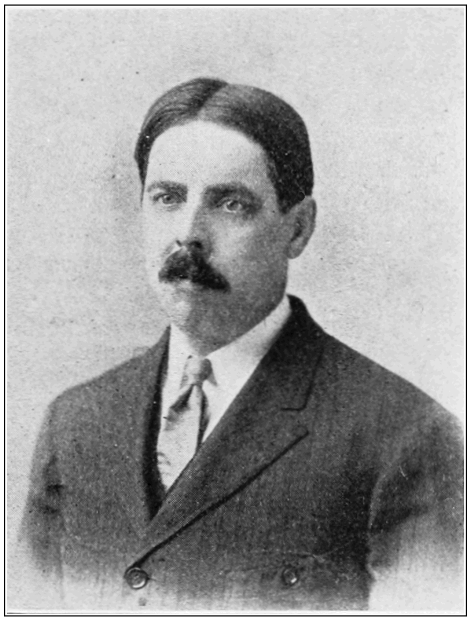


Figure 2. Edward Lee Thorndike, American psychologist, developed the "Law of Effect", which describes how animals learn through trial and error. This idea went on to influence reinforcement learning in computer science. Accessed from [3].

perceived value of the states in our environment over time?

Reinforcement learning methods can help us find solutions in different scenarios similar to those mentioned above. It is a type of machine learning; in other words, programming a computer to learn how to do a task without explicitly telling it what to do. However reinforcement learning is different from other types of machine learning like supervised learning and unsupervised learning. Supervised learning involves learning generalizations from data with labels, and unsupervised learning learns patterns from unlabeled data. In contrast, reinforcement learning learns from experience to achieve some type of goal. Richard Sutton and Andrew Barto explain, "Of all the forms of machine learning, reinforcement learning is the closest to the kind of learning that humans and other animals do, and many of the core algorithms of reinforcement learning were originally inspired by biological learning systems" [2] (See Section 1.1).

So what are the essential parts of a reinforcement learning framework? In reinforcement learning there is the agent, which makes decisions, a policy (defines how the agent behaves), a reward signal, a value function, and sometimes a model of the environment. A reinforcement learning agent seeks to maximize the value of the states it visits over time. If we understand how our environment transitions from state to state, we can use this model to plan ahead and explore the value of future states. Reward looks at the immediate utility from entering a state. Value is different than a reward signal because it is a long term look at all the rewards gained from a state in the future [2] (See Section 1.3). Refer to Figure 1 for a visual depiction of reward, state, and action.

The goal of this paper is not necessarily to talk about the latest and greatest achievements in reinforcement learning, because those can be found in current research papers on this topic. Instead, we will explore the connections, the ideas, and the people that gave rise to reinforcement learning. We start with the early beginnings and background of reinforcement learning as well as key definitions that will be helpful to the reader. We then talk about approaches to the problem of finding the value of a state and then learning a policy directly (as well as a combination of both of these so-called actor-critic methods). Next, we will introduce two difficult problems in reinforcement learning: exploration vs. exploitation tradeoff and the credit assignment problem. We do not explore solutions to these problems; the goal is to make the reader aware of these issues which are found in a wide variety of reinforcement learning problems.

Early Beginnings and Background

There are multiple ideas that combine to create modern reinforcement learning as we see it today. In this section we will briefly talk about two threads that influenced the main ideas discussed later. These are trial and error learning and Markov Decision Processes. We will then define important terms such as model-free, model-based, online, and offline in the context of reinforcement learning.

Since the conception of artificial intelligence, researchers have tried to have tried to mimic the results found in animal psychology [2] (See Section 1.7). For instance, many animals learn by trial and error. Consider a dog learning how to do a trick, like rolling over. It might take a while for the dog to complete the action the first time, but after the dog rolls over the trainer will likely give the dog a treat. The treat is a stimulus that reinforces the dog's behavior.

In 1911, Edward Thorndike, an American psychologist, published *Animal Intelligence* which proposed that animals learned not by reasoning or imitation, but through the reinforcing of good behaviors through trial and error. He came up with this conclusion after experiments on cats and dogs and later on fish and monkeys [4]. He developed what is now known as the "Law of Effect" which says that if a behavior is followed by satisfaction (or a reward), that behavior is more likely to occur. However, if a behavior is followed by discomfort, the connections to that behavior will be weakened [5]. Thus by trial and error, animals learn to behave in a way that will bring pleasure instead of pain.

Another great thinker who contributed to the idea of trial and error learning was Alan Turing. Turning was a brilliant mathematician who helped break Nazi ciphers during World War 2. In a seminal paper in 1936 he proved that mathematics will always have undecidable propositions. He is regarded for developing important underlying research in computer science and artificial intelligence [6]. In a 1948 paper, Alan proposed a machine that had a pleasure-pain design that follows the "Law of Effect." In this system, random choices for missing data are made if we do not know what to do in a certain situation. These temporary entries are used until a stimulus is received, and in the case of a pain stimulus, all tentative entries are cancelled. However, if a pleasure stimulus arrives the tentative entries are made permanent [7].

With a basis for solving the problems based in psychology, we need a framework to set up our decision making problem. This is typically done with the Markov Decision Process (MDP). Ron Howard is an important contributor in this area (and the dynamic programming methods discussed later). When Howard was at MIT he went to the physics department and just happened to drop in on a talk by Stanislaw Ulam, a physicist who worked on the Manhattan Project. Ulam talked about a framework called Markov Processes, which Howard found interesting. Howard ended up doing graduate research on the topic and contributed significantly to the research on MDPs which are a specific type of Markov Process [8].

In his book [9] (See Chapter 1), Howard gives the example of a frog in a lily pond as a way to describe a Markov Process. As time goes by, the frog jumps from one lily pad to another. The state of the system is the number of the pad occupied by the frog, the state transition is the frog's leap.

If we add rewards and decisions into this analogy, we have a Markov Decision Process (MDP). Defined by Professor David Silver [10] (See Lecture 2), a Markov Decision Process is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where

- $\mathcal{S}$ is a set of states
- $\mathcal{A}$ is a set of actions
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}^a_{ss'} = \mathbf{P}[S_{t+1} = s' | S_t = s, A_t = a]$
- $\mathcal{R}$ is a reward function, $R^a_s = \mathbf{E}[R_{t+1} | S_t = s, A_t = a]$
- γ is a discount factor $\gamma \in [0, 1]$

States in a Markov Decision Process have what is called the Markov Property:

$$\mathbf{P}[S_{t+1} | S_t] = \mathbf{P}[S_{t+1} | S_1, ..., S_t]. \qquad (1)$$

In other words, the current state captures all the information from the history of the states we have visited. Here the states are fully observable. For the sake of simplicity, we will not explore how to learn in partially observable states, even though these are more similar to many real world scenarios. What is interesting about Markov Decision Processes is that they can define almost any fully observable system that an agent can interact with. For example, they can be used for zero-sum games where we model the opponent's actions in our transition probability matrix.

These two key ideas play an integral role in the reinforcement learning methods talked about in the next sections. Trial and error learning can be thought of as the "reinforcement" in reinforcement learning. Many of the learning methods discussed later follow the idea of reinforcing behaviors that give a high return on reward. Markov Decision Processes are important because they form the common framework for most reinforcement learning problems. In later sections the reader may want to refer to the definition of an MDP described above to understand each part of the equations presented.

Before we move to the next section we briefly define a few more terms that will be of use to the reader. These include model-free vs. model-based methods and online vs. offline reinforcement learning. Reinforcement learning algorithms can typically be divided into model-free and model-based approaches. Model-free methods seek to learn how to interact in an environment without a model of the environment. One example of this is a drone flying in weather patterns it cannot predict. Model-based methods on the other hand seek to learn (or are given) a model of the environment as they learn how to act [11]. A computer program learning how to play the game of chess (after it has learned the rules) would be an example of this. Typically model-free methods have the ability to be applied to more realistic scenarios where we don't have a model of the environment. One advantage of model-based methods however, is the ability to "look ahead" and predict what will happen in the future. Many of the value-based methods mentioned later (dynamic programming and monte-carlo learning) require a model of the environment. Some temporal difference methods (like Q-learning) as well as many policy-gradient approaches are not constrained to have model.

Next, we will denote the difference between online and offline reinforcement learning. Many of the reinforcement learning agents that have been implemented in the real world are online agents. They observe the environment state and make a decision, observing the consequences of their choices. They can then adapt and respond to the feedback they receive the reward stimulus. This is what most of the methods in this paper are applied to. Offline (or batch) reinforcement learning seeks to take offline data (sequences of actions and states from another agent) and learn a policy based on that information. This is a more difficult problem because there is no exploration for an offline reinforcement learning agent; it has to infer how to act based on another agent's experience [12].

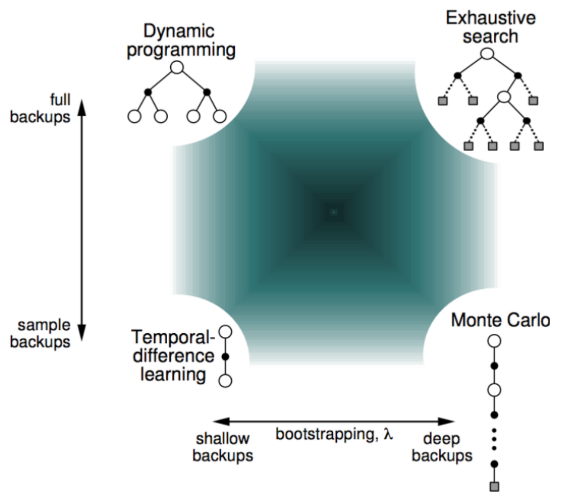


Figure 3. The Unified View of Reinforcement Learning. Here the ideas of temporal difference learning, Monte Carlo learning, and dynamic programming can be viewed on a continuum. By varying the amount of bootstrapping and backups, we can arrive at any of these methods. With full and deep backups we have an exhaustive search, which has high computational and spatial complexity. Accessed from [13].

Value-Based Learning

How do we learn the value of states we end up in? This is one of the key questions in reinforcement learning. If we understand the value of the state we are in, we can look ahead at values of the next states and come up with a decision about the next action we should take (a policy).

The main ways we can come up with an accurate value function are bootstrapping and sampling. Bootstrapping is using our own estimates of the value of future states to update the value of the current state. Sampling looks at the value of many future returns (the accumulation of rewards over time) [10] (See Lecture 4). We can put some of the future ideas we will discuss (dynamic programming, temporal difference learning, and Monte Carlo learning) in this framework to give us the unified view of reinforcement learning (see Figure 3).

Dynamic Programming

Dynamic programming can be thought of as backward induction. Basically, we find a solution to the smaller (or simpler) problems and work backwards to solve the entire problem. This idea to solve sequential decision problems with backward induction was used back in the 1940s with Jon Von Neumann and Morgenstern and applied to game theory. In sequential decision making, these "simpler problems" that we solve first could include the value of states at the end of the episodes. We then can work backwards and update the values of earlier states. In a 1949 paper by Arrow, Blackwell and Girshick, they solved a statistical decision problem in a way that used dynamic programming as we see it today. Richard Bellman is given credit for showing that backward induction can solve a huge class of sequential decision processes [14]. Bellman came up with the name "dynamic programming" because he wanted to hide the fact that he was doing research from the Secretary of Defense (who didn't like the word "research"). He knew that dynamic programming was a name that "not even a Congressman could object to" [15]. This leads us to the Bellman expectation equation (named after Richard Bellman).

Before exploring this equation, we will define two specific types of functions that will be used throughout the paper. We continue to use the notation from David Silver [10] (See Lecture 3), and will do this for the rest of the paper. The first is the **state-value function** $v_\pi(s)$. This is the expected return of an MDP starting from state s and then following a policy π:

$$v_\pi(s) = \mathbf{E}_\pi[G_t | S_t = s]. \tag{2}$$

Here G_t is the return, or the sum of discounted rewards following a specific trajectory through our state space. Specifically,

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \tag{3}$$

where γ is the discount factor defined in the MDP description. Notice that there may be stochastic transitions built into our environment, so the outcome of following the same policy will not always give us the same result. That is why we need to take the expectation of all of these trajectories in our state space.

The second function is the **action-value function** $q_\pi(s, a)$. This is the expected return starting from state s and taking action a, and then following policy π. In other words:

$$q_\pi(s, a) = \mathbf{E}_\pi[G_t | S_t = s, A_t = a]. \tag{4}$$

For the state-value function we have the following relationship (called the Bellman expectation equation):

$$v_\pi(s) = \mathbf{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]. \tag{5}$$

We can also create a version of the Bellman expectation equation from the action-value function by substituting G_t with the expected reward added to the discounted value of the next state.

Once we have calculated values of a future state S_{t+1} we can compute the value of our current state S_t. If we have an idea of the state transitions, we can visit every state in our MDP and iteratively update our value function (state-value or action-value). We will now briefly highlight two methods, policy iteration and value iteration, that learn the state-value function over iterative updates.

We use the Bellman expectation equation to do policy iteration. **Policy iteration** is broken up into two parts: policy evaluation and policy improvement. With policy evaluation we take a policy π (which could be random to start) and find the value function from the policy:

$$\begin{aligned} v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) (\mathcal{R}_s^a \\ + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s')). \end{aligned} \tag{6}$$

Notice that this equation follows from the Bellman expectation equation mentioned earlier. Here we are taking an expectation of the current reward added to the discounted value of the next state we could end up in. The expectation is taken with respect to our probability distribution $\pi(a|s)$, which is our policy. This is the probability of taking action a given that we are in state s.

After we evaluate our policy, we can improve it by updating π to be π'. This is known as policy improvement. We pick π' by taking the best (or

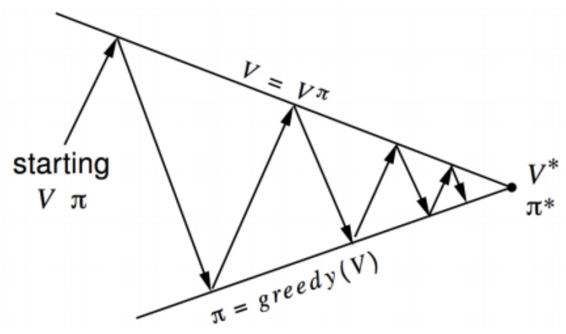


Figure 4. A visual diagram of policy iteration. In policy iteration we go back and forth between updating our current policy π (policy improvement) and determining our new value function V^π (policy evaluation). Eventually we reach the optimal policy and value function, π^* and V^*. Accessed from [16].

greedy) action in each state. By jumping back and forth between policy improvement and policy evaluation, we will converge to the optimal value function. Next, we turn our attention to **value iteration**. This is also used to approximate our value function. At each iteration and for every state in our environment we update $v_{k+1}(s)$ from $v_k(s')$ with the following equation:

$$v_{k+1}(s) = \max_{a\in\mathcal{A}}\{\mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a v_k(s')\}. \quad (7)$$

This equation follows what is known as the Principle of Optimality. According to David Silver [10] (See Lecture 3), the Principle of Optimality states that a policy achieves the optimal value from state s if and only if two conditions hold. First, we need to be able to reach state s' from s. Next, we need the policy to achieve the optimal value from state s' as well. Essentially, we can find the optimal value of our current state by looking for the optimal action right now and assuming we will take optimal actions in the future. This is the essence of dynamic programming: finding the optimal solution to smaller pieces of the problem first and then using these solutions to find the optimal solution of the entire problem.

That being said, one unavoidable question remains. How do we know that these methods eventually converge to an optimal value function? The answer lies in what is known as the Contraction Mapping Theorem.

The Contraction Mapping Theorem is also known as Banach's Fixed Point Theorem [17] and was developed by Stefan Banach. During World War 1, Banach was unfit for military service because of his poor eyesight. He worked on road construction and taught in local schools instead of fighting for his home country of Austria-Hungary. By the time the war was over, he had worked on several papers in his spare time. He went on to found a new mathematics journal and contribute innovations to functional analysis [18]. One of these was the Contraction Mapping Theorem which states:

Theorem 1:

For any metric space $\mathcal{V}$ that is complete (i.e. closed) under an operator $T(v)$, where T is a γ-contraction, then T converges to a unique fixed point at the linear convergence rate of γ.

In this case we can define the Bellman optimality backup operator,

$$T^*(v) = \max_{a\in\mathcal{A}} \mathcal{R}^a + \gamma \mathcal{P}^a v \quad (8)$$

and the Bellman expectation backup operator,

$$T^\pi(v) = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v. \quad (9)$$

These both fit the definition of an operator in the theorem.

Notice how these correspond to the Bellman equations for the value iteration (Equation 7) and policy iteration methods (Equation 6). The theorem says that each of these operators converge to a "fixed point." For the Bellman expectation backup operator (policy iteration), the fixed point is v_π, or the value function where we follow the current policy π. Then as we do policy improvement and the policy evaluation again, we inch toward the optimal value function. See Figure 4 for a visualization of this process.

For the Bellman optimality operator, the fixed point is the optimal value function. In other words, value iteration will eventually converge to the optimal value function [10] (See Lecture 3).

Figure 5. Stanislaw Ulam, one of the main contributors to Monte Carlo methods. Accessed from [20].

Monte Carlo Learning

In Los Alamos in the 1940s, the situation set to give us Monte Carlo methods. First, the problems were based on complex simulations, like the simulation of neutron histories, hydrodynamics, and thermonuclear detonation. Next, a group of talented people were brought together to work on the fission problem. Within this group were the following brilliant mathematicians: Jon von Neumann, who went on to develop digital computers, Stan Ulam, who was interested in using "statistical sampling" for many problems, and Robert Richtmyer, who ran numerical analysis activities at Los Alamos. They were joined by physicists Enrico Fermi, Nick Metropolis, and Edward Teller. Lastly, the researchers had access to human computers using hand calculators and early digital computers [19].

After Ulam came up with the idea, von Neumann saw the potential of the new approach. In a letter to Richtmyer, von Nuemann explained how the new method could be used to "generate a statistically valid picture for the genealogical history of an individual neutron" [21]. Monte Carlo methods would go on to find applications in other areas: numerical linear algebra, partial differential equations, and integral methods [19].

Another area where Monte Carlo methods are useful is in reinforcement learning. The main idea behind Monte Carlo methods is to take many statistical samples to arrive at an unbiased estimate of the value of a state. As mentioned earlier, in reinforcement learning we define the value of a state to be the expected return [2] (See Equation 2).

Recall that G_t is the return from time t. In order to estimate the value of a state, we can average the returns from a given state to converge towards the expected value. One issue with Monte Carlo learning is that we need terminating episodes so we can accumulate reward. Another issue is that there is high variance, so we need many samples to narrow down on the true value. However the big advantage of Monte Carlo Methods is that the values are unbiased; in other words, the expectation of the samples have good properties for convergence [10] (See Lecture 4).

Temporal Difference Learning

In a paper in 1988, Richard Sutton used the term temporal difference (TD) for the first time. He described how temporal difference learning works and its advantages. He gives an example of a weatherman trying to predict on each day of the week whether it will rain on the following Saturday. A conventional approach will make predictions from each day of the week about whether or not it will rain on Saturday. Once Saturday comes around, we have to update all of our previous predictions for each of the days. With TD methods, we will update each Saturday rain prediction based on the next day's prediction. This is less computationally expensive because once we have the tomorrow's prediction, we can update today's. [22]. In other words, we are bootstrapping future values to update our current ones.

There are many different variations of temporal difference learning, but they boil down to this idea of updating values by bootstrapping values from the future. For instance, we can update the value function to move towards the estimated return $R_{t+1} + \gamma V(S_{t+1})$:

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} \\ + \gamma V(S_{t+1}) - V(S_t)),$$

where α is some learning rate.

This algorithm is called TD(0). Note that we can combine the ideas from Monte Carlo methods and Temporal Difference learning by defining the n-step estimated return as:

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + ... \\ + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

We then have n-step temporal difference learning:

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^{(n)} - V(S_t)) \quad (10)$$

If we let the number of steps go out to a terminal state, we have a sample from a Monte Carlo Simulation [10] (See Lecture 4).

Another important development in the area of temporal difference learning is Q-Learning. Proved to converge by Christopher Watkins in 1992 [23], it is similar to temporal difference learning described previously, except this time we are using the action-value function instead of the state-value function. The action-value function (Q-values) are updated as follows:

$$(S, A) \leftarrow Q(S, A) + \alpha(\delta) \quad (11)$$

where

$$\delta = R + \gamma \max_{a'} Q(S', a') - Q(S, A) \quad (12)$$

Equation 11 is a good example of the integration of the different ideas for value-based methods seen throughout this section. We can see the Bellman equation being used where we are assigning $Q(S, A)$ to move towards $R + \gamma \max_{a'} Q(S', a')$. We see backwards induction from the future state S'. This is similar to the value iteration equation seen earlier. Finally, we have temporal difference learning where we assign the value to move towards the temporal difference $R + \gamma \max_{a'} Q(S', a') - Q(S, A)$ by a factor of α.

Figure 6. Richard Sutton, a key contributor to temporal difference learning and reinforcement learning. Accessed from [24].

Policy-Based Learning

When our environment becomes too complex (such as continuous states and actions in high dimensional spaces), we cannot solve the problems with value-based reinforcement learning methods. In these difficult scenarios, these learning methods don't have guarantees to converge. Sometimes we need a different approach to solve these problems. Instead of approximating the value function, we can skip to the end goal and try to approximate a policy function instead. These methods, called policy gradient methods, have advantages over value-based learning approaches. Policy gradient methods often have fewer parameters and some guarantees of convergence to a local optimum [25].

Policy gradient methods take a policy objective function $J(\theta)$ and seek to find a local maximum by ascending the gradient of the policy with respect to parameters θ:

$$\Delta\theta = \alpha \nabla_\theta J(\theta) \quad (13)$$

Here $\nabla_\theta J(\theta)$ is the policy gradient (a vector of the partial derivatives of each parameter with respect to the objective function).

In a seminal paper by Richard Sutten, David McAllester, Stinder Singh, and Yishay Mansour in 2000, the authors prove that an unbiased estimate of the gradient can indeed be obtained from experience and by using an approximate value function satisfying certain properties [26].

This Policy Gradient Theorem as stated by David Silver is as follows [10] (See Lecture 7):

Theorem 2:
For any differentiable policy $\pi_\theta(s,a)$ that correspond to the start state objective, average reward objective, or average value objective the policy gradient is:

$$\nabla_\theta J(\theta) = \mathbf{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)]$$

This is an important theorem because it basically says that as long as we can estimate the action-value function Q^{π_θ} with respect to our policy function π_θ, as well as calculate the gradient of our policy function, we can estimate the gradient of our objective function. We can choose to estimate the action-value function by averaging the returns from each state (Monte Carlo estimation), but this induces high variance. In the next section we will look at a slightly different approach.

Actor-Critic Methods

Is there a way to combine the advantages of value-based and policy-based methods? This is the exact idea behind actor-critic methods. The actor here refers to the agent following a policy and the critic is the value function that can be used to determine how effective each action is. Recall that the Policy Gradient theorem relies on an estimate of the action-value function. Instead of using returns to estimate $Q^{\pi_\theta}(s,a)$, we can parameterize the value function like we did in the value-based methods section. Thus we have a new function $Q_w(s,a)$ with parameters w such that:

$$Q_w(s,a) \approx Q^{\pi_\theta}(s,a) \tag{14}$$

Using this together with our parametrized policy function, we can estimate the policy gradient and move the policy parameters towards a local optimum [10] (See Lecture 7). By adding in the parametrized action-value function (the critic) we are able to give the actor a low-variance knowledge of its performance, speeding up the learning process [27].

Problems Encountered in Reinforcement Learning

We will now explore two important problems in reinforcement learning. These are the exploration vs. exploitation dilemma and the credit-assignment problem. There are many ways to approach and solve these problems, however this is not our emphasis here. The goal is to make the reader aware of each problem and how it relates to reinforcement learning.

Exploration vs. Exploitation

In 1991, James G. March published a seminal paper [28] about organizational behavior titled "Exploration and exploitation in organizational learning". Although not speaking about reinforcement learning (or computer science for that matter), it highlights an important problem that comes up for all decision making agents. The idea comes from this question, how do we balance exploitation (doing what we already know will work) with exploration (finding new and possibly better options)? A common example that can be seen throughout history is when to refine existing technology or when to adapt and make new technology. This idea of exploration vs. exploitation can be seen in even the basic decisions of our lives. Returning to the example from the introduction, do I go to a new restaurant (which might become my new favorite) or do I go to a restaurant that I already know I like?

This dilemma is very prevalent in reinforcement learning. In many real-world scenarios our environment is too large to model and we are required to search for high valued states. In order to be successful we need to learn a strategy to find and exploit those high valued states.

Multi-Armed Bandit A simple way to formulate this problem is in terms of what is called the Multi-Armed Bandit. Suppose we have n slot machines that may be pulled in any order. Each pull of an arm takes one time unit and we can only pull one arm at a time. Pulling an arm either results in a success or failure [29]. To simplify the problem, each slot machine gives us the same amount of money (say \$1000). We don't know the probability of success for any of the machines, but the goal is to make as much money as possible.

Figure 7. Marvin Minskey, who contributed to many of the foundational topics in artificial intelligence. Accessed from [30].

Which arms do we pull and how often do we pull them? This is a classic example of exploration (trying a slot machine we aren't sure about) vs. exploitation (sticking to the slot machine we have had the most luck with).

In the end, it comes down to our ability to quantify the value of information we are seeking. As David Silver [10] (See Lecture 9) explains, exploration is useful because it gains information. Therefore, it makes sense to explore situations we are uncertain about. However, if the information we may gain is not useful, then exploring might not be worth it. In other words, we want to know the difference between what we are willing to lose now (by exploring) and what we will gain in the long term. This will vary based on the situation: how much it costs to explore, how much we can exploit right now, and potential payoff from exploration. In the end, there is no easy answer.

The Credit-Assignment Problem

As a boy, Marvin Minskey read through his father's copies of Sigmund Freud, where he developed an interest in the human mind. While many of his counterparts used computers to solve complex numerical problems he was focused on theories about how thinking worked. He studied physics at Harvard and then earned a PhD of mathematics at Princeton. In 1956, Minskey helped organize the "Dartmouth workshop" of 1956, considered by many to be the founding event of artificial intelligence [31].

Minskey went on to publish a seminal paper in Artificial Intelligence [32] where he describes what is known as the credit-assignment problem. He illustrates the problem with an example. In a study called "Friedberg's Program-Writing Program," there is the goal to write a program for a very simple digital computer. A simple problem is defined: "compute the AND of two bits in storage and put the result in an assigned location." A device generates a random program, and the program is run to see if it is successful in completing this task. The successful information is used to reinforce individual instructions in the program. The program tries to find effective instructions independently of the other instructions. It was eventually able to solve the simple problem, but it took 1000 times longer than pure chance. Why is this the case?

Minskey argues that the problem lies with credit-assignment. Since the instructions depend on each other, the credit for a working program can only be assigned to functional groups of instructions, not to individual instructions. As another example, when a grandmaster wins a game of chess, how do we assign credit for winning to each of the moves he made? The moves are interconnected and dependent on each other, and this makes it hard to assign credit to individual actions. Minskey notes that by breaking our problem down into parts and solving them sequentially (or recursively) we can hope to have more success in dealing with credit-assignment issues.

Conclusion

In the end, reinforcement learning is made to mimic how we learn as humans. The idea is based on psychology: our brain reinforces our behaviors that give us rewards. With the theory of dynamic programming, temporal difference learning, and Monte Carlo methods, researchers built a mathematical foundation that addressed the problem of finding a value function. Other ideas were developed, like finding a policy function directly. We can combine value based and policy based approaches to get actor-critic methods. Built into reinforcement learning are two difficult problems: the exploration/exploitation dilemma and the credit-assignment problem. If these problems are any indication, reinforcement learning is still an open area of research with much to be explored.

REFERENCES

1. David Silver, "Fully observable environments." https://www.davidsilver.uk/wp-content/uploads/2020/03/intro_RL.pdf, 2015. [Online; accessed December 8, 2020].
2. R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press Ltd, 2 ed., 2018.
3. Wikipedia Commons, "Edward lee thorndike." https://commons.wikimedia.org/wiki/File:PSM_V80_D211_Edward_Lee_Thorndike.png, 2012. [Online; accessed December 8, 2020].
4. "Edward l. thorndike (1874–1949) - the man and his career, a psychology for educators, education as specific habit formation." https://education.stateuniversity.com/pages/2509/Thorndike-Edward-L-1874-1949.html, 2020.
5. E. Thorndike, *Animal Intelligence: Experimental Studies*. Animal behavior series, Macmillan, 1911.
6. "Alan turing." https://www.biography.com/scientist/alan-turing, Jul 2020.
7. A. Turing, "Intelligent machinery (1948)," *B. Jack Copeland*, p. 395, 2004.
8. Informs, "Howard, ronald a.." https://www.informs.org/Explore/History-of-O.R.-Excellence/Biographical-Profiles/Howard-Ronald-A, 2020.
9. R. A. Howard, *Dynamic programming and Markov processes*. M.I.T. Press, 1960.
10. D. Silver, "Teaching." https://www.davidsilver.uk/teaching/, Mar 2020.
11. OpenAI, "Part 2: Kinds of rl algorithms¶." https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html, journal=Part 2: Kinds of RL Algorithms - Spinning Up documentation, 2018.
12. S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," 2020.
13. David Silver, "Unified view of reinforcement learning." https://www.davidsilver.uk/wp-content/uploads/2020/03/MC-TD.pdf, 2015. [Online; accessed December 8, 2020].
14. J. Rust, "Dynamic programming," *The New Palgrave Dictionary of Economics*, vol. 1, p. 8, 2008.
15. S. Dreyfus, "Richard bellman on the birth of dynamic programming," *Operations Research*, vol. 50, no. 1, pp. 48–51, 2002.
16. David Silver, "Policy iteration." https://www.davidsilver.uk/wp-content/uploads/2020/03/DP.pdf, 2015. [Online; accessed December 8, 2020].
17. "Banach fixed point theorem." http://wiki.gis.com/wiki/index.php/Banach_fixed_point_theorem, 2011.
18. T. E. of Encyclopedia Brittanica, "Stefan banach." https://www.britannica.com/biography/Stefan-Banach, 2020.
19. M. Mascagni, "Monte carlo methods: Early history and the basics."
20. Los Alamos National Laboratory, "Stanislaw ulam." https://commons.wikimedia.org/wiki/File:Stanislaw_Ulam.tif, 2015. [Online; accessed December 8, 2020].
21. N. Metropolis, "The beginning," *Los Alamos Science*, vol. 15, pp. 125–130, 1987.
22. R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.
23. C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.
24. Steve Jurvetson, "Richard sutton.." https://commons.wikimedia.org/wiki/File:Richard_Sutton,_October_27,_2016.jpg, 2016. [Online; accessed December 8, 2020].
25. J. Peters, "Policy gradient methods." http://www.scholarpedia.org/article/Policy_gradient_methods, 2013.
26. R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, pp. 1057–1063, 2000.
27. I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.
28. J. G. March, "Exploration and exploitation in organizational learning," *Organization science*, vol. 2, no. 1, pp. 71–87, 1991.
29. J. C. Gittins, "Bandit processes and dynamic allocation indices," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 41, no. 2, pp. 148–164, 1979.
30. Wikipedia, the free encyclopedia, "Marvin minsky was visiting the olpc offices and picked up a firefox wrist band.." https://commons.wikimedia.org/wiki/File:Marvin_Minsky_at_OLPCb.jpg, 2008. [Online; accessed December 8, 2020].

31. A. Conner-Simons and R. Gordon, “How three giants of early computing got their ides and got them across,” *MIT Technology Review*, pp. 13–14, 2019.
32. M. Minsky, “Steps toward artificial intelligence,” *Proceedings of the IRE*, vol. 49, no. 1, pp. 8–30, 1961.

Jamison Moody is a Computer Science Masters student and Brigham Young University, where he earned a B.S. in Mathematics with an emphasis in Applied and Computational Mathematics. He is interested in fundamental research in reinforcement learning and deep learning, particularly in offline RL. He is fascinated with the potential that reinforcement learning has to help us make decisions in the real world. When he is not in school you can find him inventing games and playing basketball.

Long Short-Term Memory Recurrent Neural Networks in Artificial Intelligence

Gabriel Smith
Brigham Young University

***Abstract*—Machine learning, deep learning and artificial intelligence continue to push the world towards greater innovations and opportunities. An overhyped artificial intelligence history rich in trial, error and disappointment found major breakthroughs by increases in computational power and discoveries in mathematical analysis. We give a brief history of neural networks, both biological and artificial, and how scientists discovered these insights in neurons passing information between each other. We review the mathematical structure of artificial neural networks, including an overview and history of significant mathematical breakthroughs such as gradient descent and backpropagation. After building a basic mathematical understanding of artificial neural networks, we then give a brief history and mathematical overview of recurrent neural networks (RNN), along with some of its applications. We finally glaze over the history and mathematical rigor of long-short term memory (LSTM) RNNs. We finish the paper reviewing some of the major breakthroughs in science and technology that have utilized LSTM RNNs, and briefly discuss potential models that could replace LSTM RNN.**

■ SINCE THE BEGINNING OF TIME, humans have attempted to replicate and mechanize human thought. Philosophers around the world strove to create structured methods to deductive reasoning dating back to the first millennium BCE including philosophers such as Aristotle and Euclid [1]. If intelligence and understanding could be replicated, societies could progress faster and farther than anyone else. Breakthroughs in understanding and utilizing artificial intelligence would create the greatest nations, civilizations and societies to ever exist on earth. Many existing processes for humans could be easily supported and accomplished by machines, where machines could often outperform humans in various settings such as steel creation or computing mathematical solutions. That being said, if consistent breakthroughs in artificial intelligence could continue to occur, could machines one day outperform humans entirely? While skeptics generally worry about the potential power involved with understanding human reasoning, scientists and philosophers alike continue to explore general possibilities and understanding of artificial intelligence.

While scientists wanted to find greater applications and advancements in artificial intelligence, it wasn't until Dartmouth held the first AI conference that artificial intelligence received its immortalized name, showcasing to the entire world its focus and mission [3]. Despite increasing optimism about the potential and future for the newly created field [4], AI struggled to really innovate and progress due to limited computing power [5] and machines struggling to understand commonsense knowledge or reasoning [6], which essentially wiped out all funding from major

Figure 1. Humans for hundreds of years have been trying to understand and recreate intelligence. Through recent discoveries and breakthroughs in artificial intelligence, machines are replicating and outperforming humans in various areas [2]

organizations for the AI research field through the late seventies [7].

Despite another surge of funding in the early eighties from various governments including Japan and the United Kingdom [8], artificial intelligence research hit another wave of funding cuts as the market for specialized AI hardware collapsed and expectations far exceeded reality for most government funded AI research projects [9].

However, the mid nineties through the turn of the century brought great applications and advancements in specific areas of artificial intelligence in outperforming the world leading experts in their domain. Computers were beating reigning world champions in chess and jeopardy [10], while robots were autonomously driving vehicles [11]. Computer power continued to grow rapidly, where according to Moore's law, the number of transistors in a dense integrated circuit doubles every two years, thus bringing on rapid increases in computational power and memory [12]. Despite these amazing applications, artificial intelligence was hardly used or termed as such. Scientists used terms such as informatics, computational intelligence or knowledge-based systems to avoid the term artificial intelligence "for fear of being viewed as wild-eyed dreamers" [13]. Artificial intelligence had only failed to live up to its expectation and hype, thus causing a decrease in the term and overall focus of it in society.

Despite the decreased hype around artificial intelligence, computing power and memory increases have brought immense opportunities. Organizations could store increasingly more data and computers contained increasingly more computational power and speed, thus allowing significantly greater breakthroughs in artificial intelligence. Advances in deep learning, particularly recurrent neural networks, have driven progress in various areas and industries, allowing near-perfect image recognition, text analysis and speech recognition [14]. These innovations have allowed scientists to develop greater models to improve accuracy and machine understanding, which brought on the creation of the long short-term memory (LSTM) recurrent neural network (RNN).

The history of artificial intelligence has been one of severe backlash, disappointment and difficulty. However, more than fifty years of artificial intelligence research has culminated in incredible innovations and breakthroughs that answer some of society's greatest problems today. This article will focus on helping the reader understand both the history and design of LSTM RNNs by first explaining the history properties of artificial neural networks and RNNs so the reader may better understand LSTM RNNs, then show many major, existing applications of LSTM RNNs.

The Neural Network Discovery

Despite more recent innovations and applications, neural networks have been discovered since the late-19th century. Two psychologists, Alexander Bain and William James, each independently discovered much of the basic principles behind contemporary neural networks. Bain proposed that every human activity results in the firing of a certain set of neurons between each other [15]. Thus the consistent repetition of these activities strengthens these neurons to help form memory. Similarly, James suggested that activities and memories resulted instead from electrical currents flowing among the neurons. His theory proposed that individual neurons weren't necessary for every memory or action, but rather that human thought was like a flowing stream among neurons [16]. While both these scientists had different views of neural activity, they both helped formulate the general idea of neurons and how information is passed between them.

While breakthroughs came in understanding neural activity in the late-19th century, the first computational model for neural networks wasn't discovered until the forties. Neurosientist Warren McCulloch wanted to understand the idea termed ”knowing,” or how humans came to understand and interpret information. He brought on a young scientist by the name of Walter Pitts, who had previously worked with the founder of mathematical biophysics, a new area of study focused on remodeling biology in terms of physical sciences and mathematical logic [17]. These two spent long hours collaborating to determine whether the nervous system could be considered as a universal computing device. This led them to the discovery of McCulloch-Pitts neuron, the basic structure of the modern-day artificial neural network [18]. Little did these men know the impact their breakthrough would have forever on the artificial intelligence world. Despite having the basic idea and structure of the model, research really stagnated during the fifties through the turn of the century due to limited computer processing power, however renewed interest came in the late 2000s as computational power finally caught up to the artificial intelligence demand.

Biological vs Artificial Neural Networks

Integral to understanding the artificial neural networks (ANN) is understanding its similarities and differences to biological neural networks. As shown in **Figure 2**, biological neurons generally retrieve signals from dendrites, process the inputs inside the cell body then send out an output to various other neurons [19]. Our brains contain billions and billions of neurons, each simultaneously processing and sending out signals to other neurons to help us understand every thought and complete every action. These process extremely fast, often asynchronously and very efficiently, often without overworking its main host. Artificial neural networks are quite different: instead of billions of neurons, ANNs contain usually 10-1000 neurons. ANNs are extremely computationally heavy, requiring excessive amounts of energy to make these computations. However, the two neural networks are similar in their overall process: receiving, computing then transmitting information between different neurons.

While biological neurons pass information between each other, ANN neurons use math to compute the input to best predict the desired output. The goal for an artificial neural network is to learn the mathematical relationship between an input variable x and an output variable y. In its most basic sense, the network finds the weights w and constant b that best explains the relationship between x and y. Artificial neural networks generally follow the mathematical logic and pattern as shown in **Figure 3** [20]. We start by multiplying our inputs x of size n with our weights w of size n:

$$x * w = (x_1 * w_1) + ... + (x_n * w_n) \quad (1)$$

We then add in our constant b to create our equation we initialize as z:

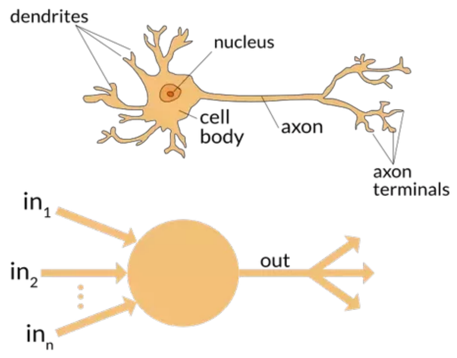


Figure 2. As shown above, biological neural networks work through the neuron receiving signals through dendrites, cell bodies processing them, then axons sending signals to other neurons. Artificial neural networks work similarly through retrieving inputs, sending inputs to the hidden layer, processing the inputs in the hidden layer then finally sending the output to the user [19]

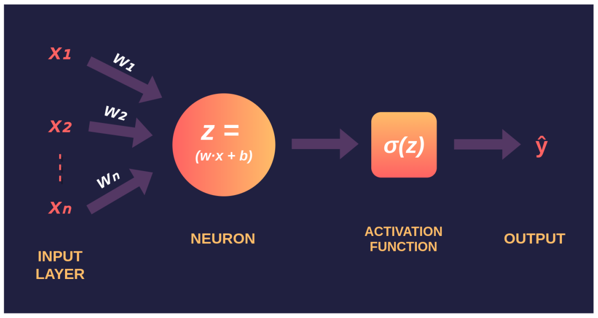


Figure 3. Basic mathematical and theoretical structure for an artificial neural network [20]

$$z = x * w + b \tag{2}$$

We can then plug our z value into our sigmoid function $\sigma(z)$ that maps our z value to a specific output $\hat{y}$:

$$\hat{y} = \sigma(z) = \frac{1}{1+\epsilon^{-z}} \tag{3}$$

The neural network uses these three equations to create a function that most accurately predicts $\hat{y}$ to the true output y. This is the mathematical basis of most neural network models; creating a function to mathematically explain the relationship between inputs x and output y. Now, we would be extremely lucky if we initially knew the correct weights w and constant b that most accurately relates x to y. We then have the question: how does the model determine the appropriate weights w and constant b to best explain x to y? That is where gradient descent and backpropagation come in, which are both integral to creating the most accurate weights w and constant b.

Optimizing the ANN using Gradient Descent with Backpropogation

While gradient descent and backpropagation are actually not part of the LSTM RNN, these are essential for understanding the ANN and RNN, which rely heavily on these two mathematical concepts. The LSTM RNN was actually primarily created to combat the problems that gradient descent and backpropagation cause for the ANN and RNN [21]. Thus to better understand the value of the LSTM RNN, we will give the overall background and theory behind these two concepts. We first give background and explain gradient descent, which will help us understand backpropagation and why gradient descent uses it.

Gradient descent was originally discovered by the famous French mathematician Augustin-Louis Cauchy in 1847, long before even neurons were discovered or neural activity was understood. He was entirely interested in minimizing a non-negative function, and found that incrementally moving the inputs in the direction of their partial derivatives would give the minimum value once the partial derivatives were 0 [22]. As we see in

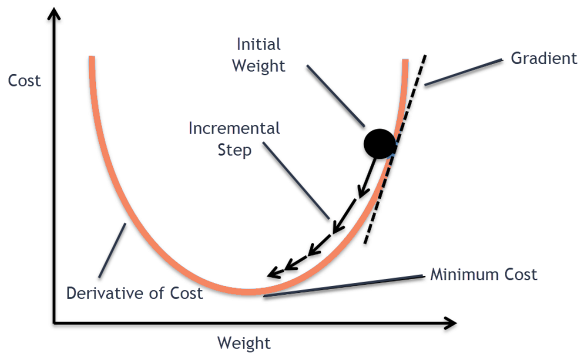


Figure 4. The basic idea of gradient descent is to gradually follow the direction of the gradient (or partial derivative) of the weight until its partial derivative reaches 0, which means the cost function is at a minimum [23]

Figure 4, the model starts with an initial weight then incrementally moves in the direction of the gradient until it reaches its minimum value [23]. While the minimum does occur for nonnegative functions, the size of the incremental steps can either lead to the optimal weight or diverge from the minimum value.

For a given weight w, we can compute its gradient, or partial derivative, in conjunction with its cost function C, then update the the weight w by the direction of its partial derivative multiplied by the learning rate α:

$$w = w - (\alpha * \frac{\partial C}{\partial w}) \tag{4}$$

Thus by incrementally moving the weight w, we can find the optimal value that helps minimize the error between our predicted output $\hat{y}$ and the actual output y. Now, we use backpropagation to compute the partial derivative of C with respect to each of the weights w and constant b.

Backpropagation became wildly popular in the early 2010s due to the increased value in modeling using neural networks, however its application to neural networks began in the mid-eighties. Backpropagation is based entirely on the chain rule in calculus, which we will review here. As an example, we let f and y be any function based on input x and let F be a function of both f and y:

$$F(f, y) = f(x)y(x) \tag{5}$$

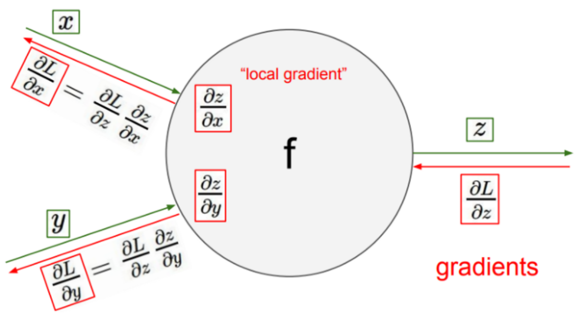

x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

Figure 5. Backpropagation allows the model to work backward through the ANN by multiplying partial derivatives together until it reaches its desired layer, thus computing the gradient to help optimize the weights w and constant b [25]

The chain rule in calculus tells us that we can compute the partial derivative of F with respect to x by computing several partial derivatives [24] until we arrive at functions containing x as shown below:

$$\frac{\partial F}{\partial x} = \frac{\partial F}{\partial f} * \frac{\partial f}{\partial z} + \frac{\partial F}{\partial y} * \frac{\partial y}{\partial z} \quad (6)$$

It was during the mid-eighties that mathematicians David Rumelhart, Geoffrey Hinton and Ronald Williams experimented with backpropagation in artificial neural networks. They found that backpropagation could generate useful interpretations of incoming data in hidden layers of neural networks [26]. Backpropagation would allow networks to compute the gradient of the weights quickly and simply to optimize the cost function and predict the outputs most accurately.

In returning to our ANN based on equations (1), (2) and (3), we can use gradient descent with backpropagation to compute the relationship between inputs x and outputs y most accurately. We first define our cost function C that helps us penalize outputs incorrectly predicted by our weights w and constant b:

$$C = \frac{1}{n}\sum_{j=1}^{n}(y_i - \hat{y}_i)^2 \quad (7)$$

Now we can use gradient descent with backpropagation to find the optimal weights w and constant b. Gradient descent allows us to incrementally find w and b by subtracting its the partial derivative of the given variable. Now, we can use backpropagation to find the partial derivatives of the cost function C in terms of our desired variables w and b. We first compute the partial derivative of C with respect to each weight w_i by backpropagation using the chain rule:

$$\frac{\partial C}{\partial w_i} = \frac{\partial C}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z} * \frac{\partial z}{\partial w_i} \quad (8)$$

We then compute the partial derivative of C with respect to the constant b:

$$\frac{\partial C}{\partial b} = \frac{\partial C}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z} * \frac{\partial z}{\partial b} \quad (9)$$

Now that we have our gradients, we can use gradient descent to incrementally move each of our desired variables towards their optimal value. We control the overall change to the variables w and b each iteration through the learning rate α. Thus we can update each weight w_i and the constant b as follows:

$$w_i = w_i - (\alpha * \frac{\partial C}{\partial w_i}) \quad (10)$$

$$b = b - (\alpha * \frac{\partial C}{\partial b}) \quad (11)$$

While this approach looks at generally one dimensional inputs and only one activation function, this model can be extended to n-dimensional inputs and n-activation functions. Because the neural network is so robust, it can find nonlinear mappings that before had been nearly impossible to compute [27]. Applications include areas such as pattern recognition, facial recognition, sequence recognition, e-mail spam filtering and even medical diagnosis.

Emergence of the Recurrent Neural Network

Despite its beginning success, the artificial neural network still had some major disadvantages. ANNs had difficulty understanding sequential data, as the model had no way of looking back at previous inputs. Gradients are computed after each iteration, and weights are adjusted accordingly. Thus the model couldn't check on the validity of previous inputs towards understanding the next input. For certain types of

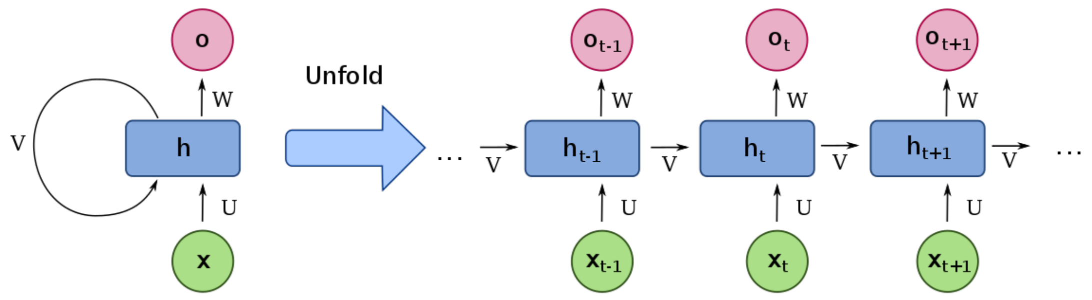


Figure 6. The Recurrent Neural Network (RNN) allows the network to learn from previous inputs to help predict subsequent outputs. It adds the previous hidden layer to the subsequent hidden layer to better understand the relationship between subsequent input data [28]

problems based on time-dependent or sequential data, models now needed the ability to check previous inputs to better compute the next output. Thus Recurrent Neural Networks (RNNs) were discovered to further help this issue.

American psychologist David Rumelhart first discovered the idea of using the memory of the model to help predict the next output [29]. While looking at the difficulties associated with backpropogation, he found that the previous hidden state could be used as a variable to help predict the next output values. Before finding the LSTM RNN, scientists Sepp Hochreiter and Jurgen Schmidhuber helped develop many of the first recurrent neural networks [30].

The RNN is very similar to the ANN, however it adds an extra input to each neuron. As seen in **Figure 6**, neurons receive both the input variables and the previous hidden layer before computing the next output, allowing the model to better understand the relationship between successive input variables. It is very similar mathematically to ANNs but adds in the previous hidden layer variable h_{t-1} with its accompanying weight v to each activation function [21]. Similar to our previous problem, we can compute our hidden layer h_t from the input x_t and previous hidden layer h_{t-1} by our σ function:

$$h_t = \sigma(u * x_t + v * h_{t-1}) \tag{12}$$

We then use a $softmax$ function to help map the hidden layer h_t to the output $\hat{y}$:

$$\hat{y}_t = softmax(w * h_t) \tag{13}$$

As we see, the RNN model increases the number of weight variables due to the model computing the weights for both the current hidden layer h_t and the previous hidden layer h_{t-1}. Computing and finding the correct weight for the previous hidden layer allows the model to understand the relationship between previous and current inputs. Thus model complexity and computation increase, however the model can significantly better understand the relationship between subsequent input variables. Our cost function remains the same from our ANN model, however we need to use gradient descent with backpropagation for each of the weights u, v and w.

Thus we use our same cost function to compute the error between the predicted $\hat{y}$ and actual output y:

$$C = \frac{1}{n}\sum_{j=1}^{n}(y_i - \hat{y}_i)^2 \tag{14}$$

We then use backpropagation to compute the gradient for each of the weights. The weight v has the following partial derivative from backpropagation, working through both the cost function C and our predicted output function $\hat{y}$:

$$\frac{\partial C}{\partial v} = \frac{\partial C}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial v} \tag{15}$$

To backpropagate through the other weights v and u, we must work through the same equations

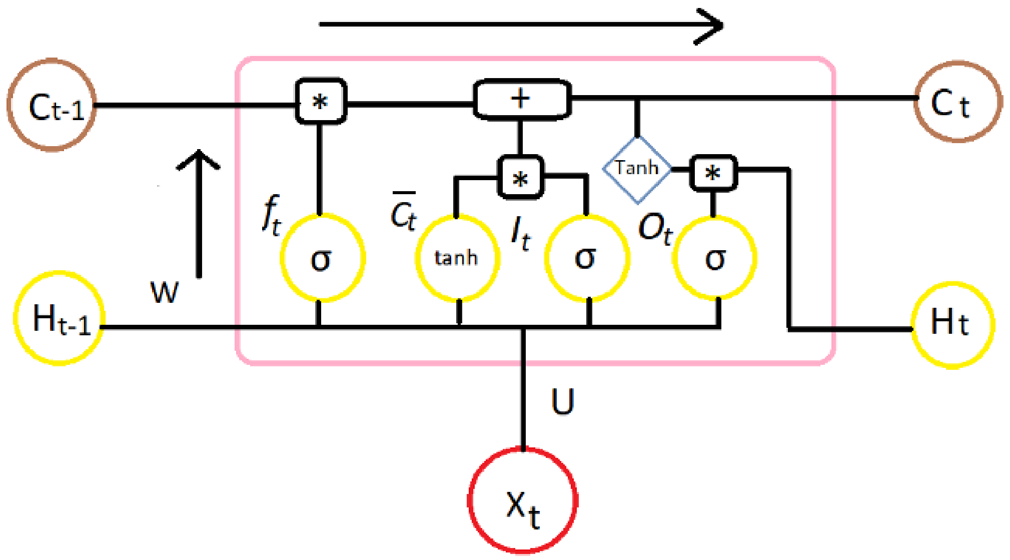


Figure 7. The Long-Short Term Memory (LSTM) RNN adds more complexity into our algorithm to help combat the gradient growing to large or too small. It utilizes the inputs x_t, previous hidden state H_{t-1} and previous memory state C_{t-1} to compute the new hidden state H_t and new memory state C_t, thus helping predict the correct values without using gradient descent with backpropagation [31]

as v but also the current hidden layer function h_t as well:

$$\frac{\partial C}{\partial w} = \frac{\partial C}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial h_t} * \frac{\partial h_t}{\partial w} \tag{16}$$

$$\frac{\partial C}{\partial u} = \frac{\partial C}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial h_t} * \frac{\partial h_t}{\partial u} \tag{17}$$

After computing the partial derivatives of C with respect to each other weights, we can use gradient descent and increment each of the weights according to their partial derivatives:

$$v = v - (\alpha * \frac{\partial C}{\partial v}) \tag{18}$$

$$w = w - (\alpha * \frac{\partial C}{\partial w}) \tag{19}$$

$$u = u - (\alpha * \frac{\partial C}{\partial u}) \tag{20}$$

These added variables and weights allowed extensive increase in the accuracy and understanding of models for sequential data. Accuracy greatly increased for various problems including time series prediction, speech recognition, time series anomaly detection, rhythm learning, music composition and protein homology detection [32] [33]. RNNs transformed how we approached sequential data, making major accuracy breakthroughs in previously difficult prediction problems.

Necessity of Long Short-Term Memory RNNs

Although RNNs increased accuracy tremendously, there were still problems that most RNNs kept running into. More specifically, both RNNs and ANNs often suffered from the exploding or vanishing gradient problem [21]. As model complexity increases, a given RNN or ANN may have anywhere from ten to one hundred different hidden layers inside of the model. Thus as we find the partial derivatives of each weight through backpropagation, partial derivatives may grow or shrink exponentially. Either problem would hurt the ability of the weights to converge to an optimal value to further minimize the cost function. Thus scientists needed a new model to find optimal weights that don't follow a linear gradient pattern, where weights could be adjusted nonlinearly. This is when scientists discovered the Long Short-Term Memory (LSTM) RNNs.

Scientists Sepp Hochreiter and Jurgan Schmidhuber in 1997 found the algorithm upon studying how to approach the vanishing gradient problem [30]. Their version included important additions to our previous RNN mathematical approach including three types of gates that pass information between the previous and current inputs but also another input called the memory state C_t. As seen in **Figure 7**, the LSTM model increases the complexity of even our RNN significantly. We will dive into the mathematical setup of the equation and its importance [31].

The model uses three different inputs into the equation: the input variable x_t, the previous hidden state H_{t-1} and the previous memory state C_{t-1} to compute four different functions, each using their own weights. The first function f is deemed the forget gate, calculated from the input variable x_t and the previous hidden layer H_{t-1} with their associated weights u_f and w_f using the σ activation function:

$$f_t = \sigma(u_f * x_t + w_f * H_{t-1}) \tag{21}$$

We then compute the candidate layer $\tilde{C}_t$ using the same inputs as the forget gate f with its own weights u_c and w_c but with the $tanh$ activation function:

$$\tilde{C}_t = tanh(u_{\tilde{c}} * x_t + w_{\tilde{c}} * H_{t-1}) \tag{22}$$

The next two functions are the input gate I_t and output gate o_t, which follow similar computation as that of the forget gate function f_t but with their own weights:

$$I_t = \sigma(u_i * x_t + w_i * H_{t-1}) \tag{23}$$

$$o_t = \sigma(u_o * x_t + w_o * H_{t-1}) \tag{24}$$

The new memory state C_t is computed using the forget gate f_t multiplied by the previous memory state C_{t-1}. The forget gate f_t values are either 0 or 1, thus it only allows important information to pass from the previous memory C_{t-1} to the new memory state C_t. The C_t computation also utilizes the input gate I_t and candidate layer $\tilde{C}_t$:

$$C_t = f_t * C_{t-1} + I_t * \tilde{C}_t \tag{25}$$

Once we have the new memory state C_t, we can compute the new hidden state H_t using the output gate o_t:

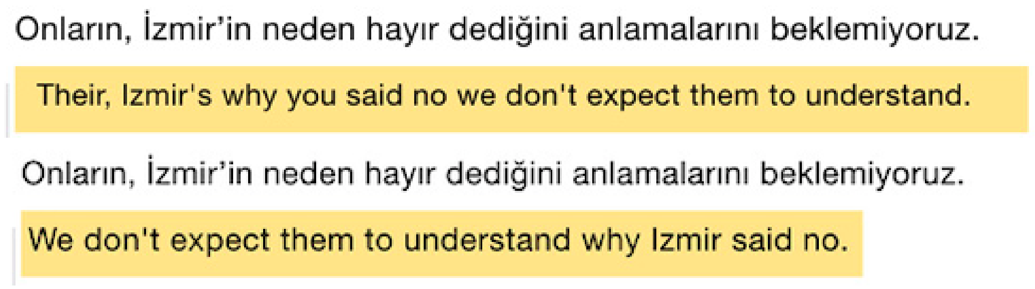
$$H_t = o_t * tanh(C_t) \tag{26}$$

We drastically increased the number of weights, parameters and inputs from our previous method, however it helps us to better understand the inputs we are looking at. The model is able to truly understand the relationship between each subsequent input variable [31].

LSTM Applications and Model Improvements

While ANNs and RNNs worked well for basic training, LSTM RNNs consistently outperformed the previous models significantly for any type of sequential data. Due to its inability to rely on the gradients themselves and its ability to forget given inputs, LSTMs won various competitions in the late 2000s. The 2009 ICDAR connected handwriting recognition competition had 10 teams from around the world use or create algorithms to most accurately recognize cursive handwriting. Three different datasets were created and tested, where each dataset differed in the number of words and training size. Amazingly, the LSTM RNN created by Alex Graves predicted each dataset with the highest accuracy, and significantly more accurate than any of the other models [35]. This is one of many examples of the LSTM significantly outperforming other models, increasing its use into various other fields.

Today many of the major tech firms using varying speech recognition and natural language processing applications are implementing LSTM models to increase the overall accuracy of their results. Google started using LSTM models for speech recognition on Google Voice, cutting transcription errors by 49% overall [36]. Facebook changed from a phonetic-based model to an LSTM RNN in 2018, which provided significantly higher accuracy and coherency as seen in **Figure 8**. Today, Facebook's LSTM RNN performs around 4.5 billion automatic translations every day [34]. And most importantly, Apple


Onların, İzmir'in neden hayır dediğini anlamalarını beklemiyoruz.

Their, Izmir's why you said no we don't expect them to understand.

Onların, İzmir'in neden hayır dediğini anlamalarını beklemiyoruz.

We don't expect them to understand why Izmir said no.

Figure 8. Facebook changed from a phonetic-based model to a LSTM RNN model to help its translation between various languages. We can see the major difference between the top and bottom translations: the LSTM RNN model was significantly more coherent and accurate [34]

started using LSTM RNNs in 2016 for quicktype in the iPhone and in Siri [37]. For Apple iPhone users, quicktype provides word suggestions that pop up once the use starts typing. The LSTM is trained on the exact data the user inputs, learning user sentence patterns to help predict what the user wants to say. I use this on a consistent basis, and the model continues to improve the more I type into it and use the suggestions.

While LSTM RNNs have proven very consistent and reliable, other models have grown in popularity over the past couple years. More recently, models called Transformers are consistently being utilized and often outperforming the LSTM RNN [38]. Transformers doesn't use any recurrent steps, but instead uses attention, where the model decides which aspects of the input are most important in the model. It does this through encoding the important aspects of the input then decoding these aspects into terms of the output. While further iterations and changes to these Transformer models may one day outperform LSTM RNNs entirely, the LSTM RNN still continues to show its value in understanding sequential data, as we see in its various natural language applications.

Conclusion

As we have seen, LSTM RNNs prove to be one of the most power models that exist today. Hundreds of years of artificial intelligence research have helped culminate into this great and useful algorithm. Now, LSTM RNNs won't be a fix all solution to any machine learning situation; often basic machine learning methods will prove adequate enough in achieving our various goals. Additionally as computation power continues to increase, LSTM RNNs may become more obsolete and useless, but the power and innovation they have created thus far will forever be respected and cherished. Artificial intelligence continues to progress, evolve and change, and while models will change and evolve, many of these general principles and mathematical discoveries will continue to play huge roles in future model iterations and breakthroughs.

ACKNOWLEDGEMENT

I would like to thank Miki Jauhiainen for reviewing and providing constructive feedback regarding this paper.

■ REFERENCES

1. D. Berlinski, *The Advent of the Algorithm*. Harcourt Books, 2000.
2. L. A. Krukrubo, "Understanding artificial intelligence," *Towards AI*, 2020.
3. A. Kaplan and M. Haenlein, "Siri, siri, in my hand: Who's the fairest in the land? on the interpretations, illustrations, and implications of artificial intelligence," *Business Horizons*, no. 62, pp. 15–25, 2019.
4. P. McCorduck, *Machines Who Think*. A. K. Peters, Ltd., 2004.
5. H. Moravec, "The role of raw power in intelligence," in *iccv*, 1976.
6. D. Lenat and R. V. Guha, *Building Large Knowledge-Based Systems*. Addison-Wesley, 1989.
7. S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Upper Saddle River, New Jersey: Prentice Hall, 2 ed., 2003.

8. H. Newquist, *The Brain Makers: Genius, Ego, And Greed in the Quest For Machines That Think*. Macmillan/SAMS, 1994.
9. D. Crevier, *AI: The Tumultuous Search for Artificial Intelligence*. BasicBooks, 1993.
10. J. Markoff, "On 'jeopardy!' watson win is all but trivial," 2011.
11. M. Belfiore, "Carnegie takes first in darpa's urban challenge," *Wired*, 2007.
12. G. E. Moore, "Cramming more components onto integrated circuits," *Electronics Magazine*, 1965.
13. J. Markoff, "Behind artificial intelligence, a squadron of bright real people," 2005.
14. Y. LeCun, Y. Bengio, and G. Hinton, *Deep Learning*. Nature, 2015.
15. A. Bain, *Mind and Body: The Theories of Their Relation*. D. Appleton and Company, 1873.
16. W. James, *The Principles of Psychology*. H. Holt and Company, 1890.
17. F. Conway and J. Siegelman, *Dark Hero of the Information Age: In Search Of Norbert Wiener–Father of Cybernetics*. Basic Books, 2005.
18. W. McCulloch and W. Pitts, "A logical calculus of ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, no. 5, pp. 115–133, 1943.
19. R. Nagyfi, "The differences between artificial and biological neural networks.," 2018. https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7#:~:text=So%20unlike%20biological%20neurons%2C%20artificial,strength%20of%20these%20signals%20varies.
20. S. K. Dasaradh, "A gentle introduction to math behind neural networks.," 2020. https://towardsdatascience.com/introduction-to-math-behind-neural-networks-e8b60dbbdeba.
21. M. Sanjeevi, "Deepnlp - recurrent neural networks with math.," 2018. https://medium.com/deep-math-machine-learning-ai/chapter-10-deepnlp-recurrent-neural-networks-with-math-c4a6846a50a2.
22. C. Lemaréchal, "Cauchy and the gradient method," 2012. https://www.math.uni-bielefeld.de/documenta/vol-ismp/40_lemarechal-claude.pdf.
23. RekhaMolala, "The ascent of gradient descent," September 2019. https://blog.clairvoyantsoft.com/the-ascent-of-gradient-descent-23356390836f.
24. "Chain rule for derivative," June 2016. https://mathvault.ca/chain-rule-derivative/.
25. K. S. Sivamani, "Back-propagation simplified," July 2019. https://towardsdatascience.com/back-propagation-simplified-218430e21ad0.
26. D. Rumelhart, G. Hinton, and R. Williams, *Learning representations by back-propagating errors*. Nature, 1986.
27. S. A. Billings, *Nonlinear System Identification: NARMAX Methods in the Time, Frequency, and Spatio-Temporal Domains*. Wiley, 2013.
28. P. T. Perez, "Deep learning: Recurrent neural networks," October 2018. https://medium.com/deeplearningbrasilia/deep-learning-recurrent-neural-networks-f9482a24d010.
29. G. E. Williams, Ronald J.and Hinton and D. E. Rumelhart, *Learning representations by back-propagating errors*. Nature, 1986.
30. S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, no. 9, pp. 1735–1780, 1997.
31. M. Sanjeevi, "Deepnlp - lstm (long short-term memory) networks with math.," 2018. https://medium.com/deep-math-machine-learning-ai/chapter-10-1-deepnlp-lstm-long-short-term-memory-networks-with-math-21477f8e4235.
32. C. Smith, "ios 10: Siri now works in third-party apps, comes with extra ai features," *BGR*, 2016.
33. W. Vogels, "Bringing the magic of amazon ai and alexa to apps on aws. – all things distributed," June 2017. www.allthingsdistributed.com.
34. A. Sidorov, "Transitioning entirely to neural machine translation," August 2017. https://engineering.fb.com/2017/08/03/ml-applications/transitioning-entirely-to-neural-machine-translation/.
35. A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, "A novel connectionist system for unconstrained handwriting recognition," *Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, pp. 855–868, June 2009.
36. Z. Danko, "Neon prescription... or rather, new transcription for google voice," 2015. Official Google Blog.
37. C. Metz, "Apple is bringing the ai revolution to your iphone," June 2016. https://www.wired.com/2016/06/apple-bringing-ai-revolution-iphone/.
38. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," October 2018. https://www.wired.com/2016/06/apple-bringing-ai-revolution-iphone/.

Gabriel Smith is currently a graduate student in the Computer Science department at Brigham Young University. He is passionate about big data and machine learning, currently conducting research in the Human Computer Interaction lab under Dr. Michael Jones on creating models to recognize figure skating jump airtime. In his free time, he is doing something outdoors whether its skiing, boating or hiking. He also loves reading and trying his hand day trading in the stock market.

Neural Sequence Modeling

Jacob A. Stern
Brigham Young University

***Abstract*—Neural sequence modeling emerged with Rumelhart and Hinton's 1986 groundbreaking paper, "Learning representations by back propagating errors". Since then, recurrent neural networks have been synonymous with neural sequence modeling. That has remained the case until recent history, when Ashish Vaswani's 2017 paper, "Attention is all you need", introduced non-recurrent self-attention for sequence modeling, ushering in a paradigm shift for neural sequence modeling. This review will traverse the history of neural network-based sequence models, focusing on the various research developments. In doing so, it will inevitably touch on other breakthroughs that catalyzed advances in neural sequence modeling and explain associated technical concepts.**

■ **IN THE EARLY 1940'S,** Warren McCulloch and Walter Pitts sought to mimic human nervous activity by creating a mathematical unit that could be composed with many other units to compute complicated logical functions. They proposed the first "artificial neuron" - the Threshold Logic Unit (TLU) [1], which used a binary valued threshold function as the transfer unit. 15 years later, Frank Rosenblatt, a Cornell Aeronautical Laboratory psychologist, sought to understand humans' ability to perceive, think, remember, and act based on past experience. His fundamental questions were 1) How do humans detect or sense information about the natural world? 2) In what form do they remember information? and 3) How do they use information stored in memory to guide future actions and perception? In 1958 he proposed one answer to these questions, similar to the TLU: a mathematical model based on human perception, which he called "the perceptron" [2]. His perceptron allowed a broader range of weight values than the binary artifical neuron. In the ensuing decade, the perceptron stirred up a flurry of hype and research in learning theory – the first wave of modern artificial intelligence.

However, in 1967, MIT professors Marvin Minsky and Seymour Papert demonstrated some limitations of single-layer perceptrons in modeling certain order-N tasks - specifically, the inability of single-layer linear perceptrons to model an XOR function. The XOR function is an example of a parity problem of order two. Said differently, the problem is: given a sequence of length 2, classify sequences correctly according to a given rule (truth table). Minsky and Papert also posited that the extension of single-layer perceptrons to many-layer perceptrons is "sterile". They published their conclusions in "Perceptrons" [3]. This book is credited with triggering the first AI winter in the 1970s.

In 1985, David Rumelhart and Geoffrey Hinton presented an algorithm for the training of multi-layer perceptrons, called "back propagation" [4]. The idea of back-propagation existed in principle before this time [5] [6] [7], but Rumelhart and Hinton were the first to concretely demonstrate its effectiveness in training a multi-layer perceptron to learn meaningful patterns. Back-propagation involves the using the chain rule of derivatives to "propagate" error backwards through a network by computing the gradient of that error with respect to the inputs, then updating the neuron weights to reduce the error. They used this scheme to train a multi-layer perceptron to correctly classify sequences up to a length of 8. They found that a multi-layer perceptron with

N hidden units could solve a parity problem for patterns of length N. Thus, Minsky and Papert's unproven intuition against multi-layer perceptrons prevented them from seeing one solution to the order-N parity problem. In their groundbreaking work, Rumelhart and Hinton also introduced recurrent neural networks and showed that there exists an equivalent feed-forward network for every recurrent network[1].

Based on this idea, they presented a recurrent neural network architecture based on a single feedforward block that operated over a sequence, whose outputs were passed into itself "recurrently", and so on for N iterations. They trained this architecture via the principles of back propagation that they introduced in the paper. They trained the recurrent neural network to solve the shift-register problem - to shift the inputs one slot to the right. This recurrent neural network was not designed to operate on or produce sequences of variable length. Its role was simply to perform the same operation over and over: to shift the sequence by n slots through n passes through the network.

They then trained the network for a more difficult problem: learning to complete sequences. They created a synthetic dataset where each input letter decoded into two digits. In this case, at each time step, they activated the corresponding input neuron and collected a loss based on the difference between the predicted next character and the target. This network is the earliest recognizable form of the recurrent neural networks that are familiar to most researchers today. Following Rumelhart and Hinton's initial work, they formalized a recurrent neural network (RNN) architecture in which memory was built up in a hidden state that was passed through the sequence. Their work in back-propagation launched the second wave of modern artificial intelligence.

The LSTM and its variants

The next significant advance in recurrent neural networks was introduced by Sepp Hochreiter and Jürgen Schmidhuber, who observed the tendency of error gradients in RNNs to either blow up or vanish as the length of the sequence increased. To resolve this, they structured the recurrent unit to have various pathways for information travel and gates to add/remove information in intentional ways, in a new architecture they termed "Long Short-term Memory" (LSTM) [9]. It includes a "forget gate" to remove previous context information, a "remember gate" to add new information, and a context gate to determine which part of the hidden state is most useful for the next recurrent token. Based on the hidden state from the previous layers and the current input token, the network governs what information passes through the gates of the current layer (see 2). The LSTM architecture is diagrammed in the left hand side of Figure 2. The proposed LSTM was able to learn an adding task requiring memory of 1000 input tokens.

Contemporary to Hochreiter and Schmidhuber's introduction of the LSTM, Schuster and Poliwal recognized that a weakness of LSTMs (and RNNs generally) is that they are inherently unidirectional. There are times when context from both sides of a sequence can be helpful for sequence-based prediction – for example, a fill-in-the-blank task. More generally, this applies to any task that relies on an input sequence as a whole, rather than on unidirectional relationships. For these types of prediction tasks, they developed the first bidirectional RNNs, demonstrating their effectiveness over RNNs on a phoneme classification task. A few year later, Hochreiter and Schmidhuber extended bidirectional RNNs to LSTMs [10].

Interlude: AlexNet

Here we step away from our discussion of LSTMs for a moment, to place into context the whirlwind of research on recurrent networks that occured in the mid- to late-2010s. In 2012, Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton entered the Imagenet 2012 image classification challenge. Previous entrants had used relatively shallow neural network architectures (≤ 3 hidden layers). Krizhevsky trained a deep[2] convolutional neural network with 8 layers via an efficient

[1]Rumelhart and Hinton credited Minsky and Papert with this idea, but a detailed search of "Perceptrons" revealed no explicit statement of this result. The 1985 Rumelhart paper was the earliest mention of recurrent networks that the author of this review discovered.

[2]By the standards of the time. Current deep neural networks have hundreds of layers.

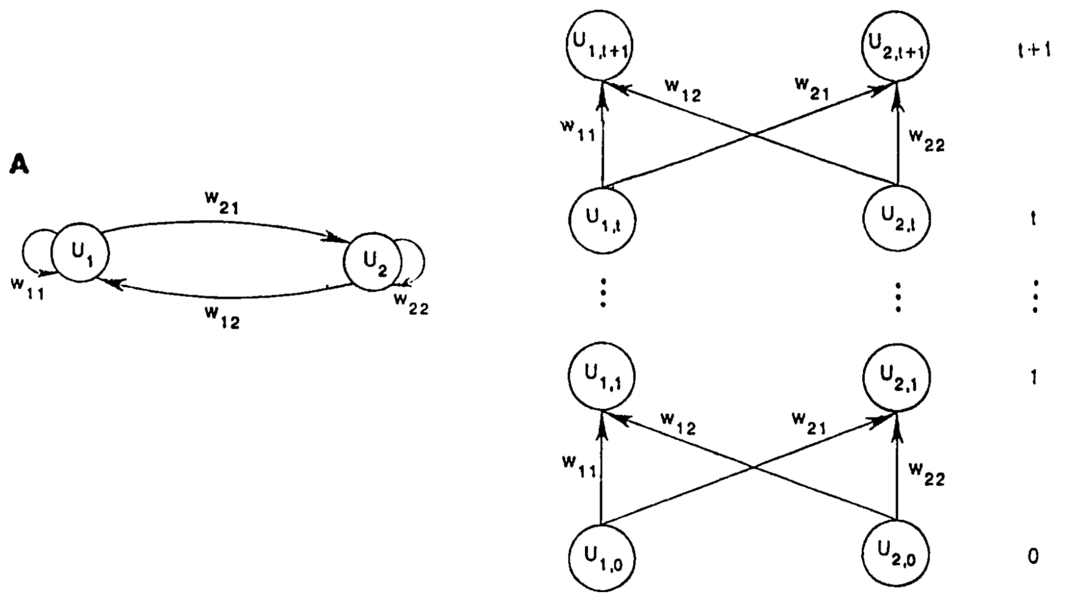


Figure 1. A diagram from Rumelhart [4] showing the how a recurrent network can be represented as a feedforward neural network.

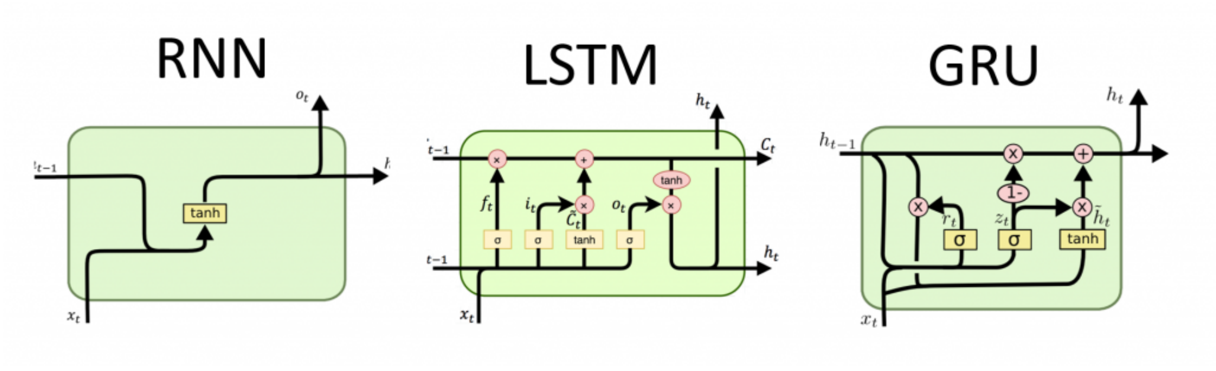


Figure 2. A comparison of a ”vanilla” RNN unit, an LSTM recurrent unit, and a GRU. Red circles represent pointwise operations, yellow boxes represent fully connected layers followed by the given activation function. Diagram from Chris Olah’s blog [8]

.

GPU implementation of the 2D convolution operation [11]. The network achieved top-5 error rate of 16 percent, compared to the next best contender achieving 26 percent top-5 error rate. The runaway success of “AlexNet”, as it became known, initiated a third wave of AI research. The success of these two ideas, 1) deep neural networks that could learn a hierarchy of features, and 2) efficient GPU implementations of neural network training, were the foundation on which the following years of research would rest.

More LSTMs and data-suited architectures

Increased interest in deep learning carried over into research in deep sequence-based neural network models. In 2014, 25 years after Hochreiter and Schmidhuber introduced the LSTM, Kyunghyun Cho et. al. introduced a simpler memory-based recurrent model, called the “gated recurrent unit” (GRU) [12]. They designed this model to work by similar principles to the LSTM, but with the forget and remember gates merged into a single unit, and with the context vector and hidden vector merged into one (see 2).

In the 2010s, neural networks were beginning to attain performance high enough to be used in practical applications. It was these applications that began to drive adaptations of network architectures and theoretical work. Whereas recurrent neural networks were originally developed in the context of learning higher-order functions, researchers now sought after RNNs for their ability to model variable-length input and output data.

One of these variable-length data tasks was machine translation, a problem in the field of natural language processing (NLP). In 2014, Ilya Sutskever (one primary authors of the AlexNet paper) wrote a paper on sequence-to-sequence language translation using LSTMs [13]. He used an encoder LSTM and a decoder LSTM to translate from English to French with high accuracy. This paper was indicative of a shift toward application-driven neural network research.

In 2015, Andrej Karpathy contributed a blog post [15] that formalized several sequence-to-sequence problems: a many-to-one mapping, a one-to-many mapping, and two types of many-to-many mappings (see Figure 3). Though not a peer reviewed publication, this post provided a simple framework for thinking about sequence modeling problems and inspired many application-driven RNN architectures. These applications included image captioning [14], caption-conditioned image generation [16], music generation and transcription [17], speech recognition [18], text-to-speech [19], and many others. This wasn't the first time that RNNs had been trained for many of these tasks, but the effectiveness of deep RNNs, similar to deep CNNs, drove a research trend of applying RNNs to everything.

CNNs for sequences

Aäron van den Oord, in 2016, introduced an alternative to RNNs for sequence modeling: causal convolutional neural networks [20] (see Figure 4). These networks resolve a key limitation of RNNs: because they operate on the entire sequence and lack sequential dependencies in the hidden layers, they can train in batches, leading to much faster convergence. In particular, they excel on very long sequences, such as audio waveforms. They trained their model ("WaveNet") to generate state-of-the-art text-to-speech.

Long-term dependencies and attention

While many researchers were experimenting with iterations on the RNN architecture and its applications, Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio were thinking about a fundamental problem with the traditional RNN training scheme. Though LSTMs had the ability to maintain a memory, they were tasked with the challenge of compressing a sequence of arbitrary length into a fixed-length vector.

To solve this problem, the authors introduced a principle for training recurrent neural network which they termed "attention" [21]. Within 5 years, attention-based neural networks would unseat RNNs in almost all major sequence processing benchmarks.

The basic principle of attention is as follows: rather than packing all the contextual information of a sequence into a fixed-length vector, store the intermediate hidden states in memory, and let the network "attend to" (pay attention to) the hidden states that it determines are most relevant for predicting the next token. There is no "remembering" or "forgetting" like there is in a traditional LSTM. The network remembers all previous tokens. The network applies a transformation to the hidden states to obtain an attention vector. If the preceding sequence has length N, then this attention vector has length N. We then multiply this attention vector by the hidden state matrix to obtain a single context vector of length d. This is the vector that we pass as the hidden state to the RNN to predict the next token.

Attention offers several benefits over traditional LSTM memory models. First, there is no temporal degradation of the context vector. By packing a sequence of arbitrary length into a fixed length vector, an RNN inevitably has to forget things that could be useful later on. On the other hand, an attention RNN stores all of these values. Second, an attention RNN allows the network to "look at" different parts of the preceding sequence when it is predicting different tokens. You could imagine that the helpful context for predicting a noun would be totally different from the helpful context for predicting a verb. An LSTM would force you to cram all of that info into one vector; attention allows the network to choose its context tokens. Third, the error

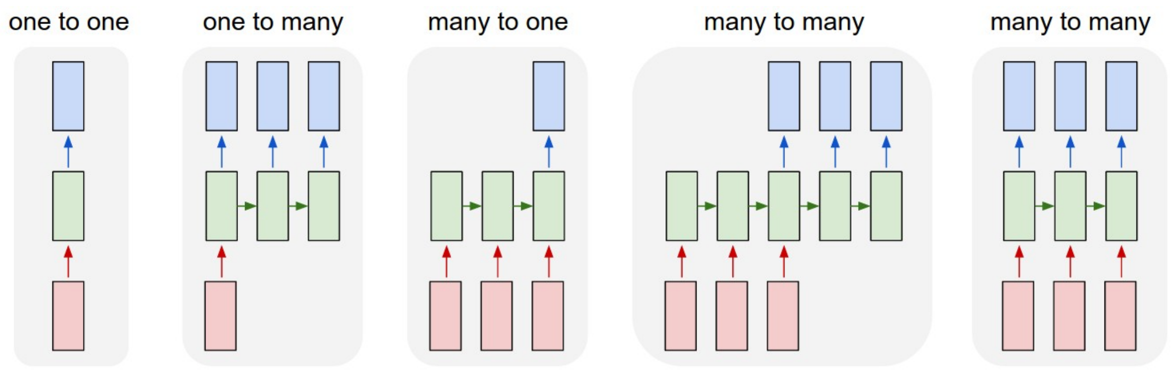


Figure 3. The sequence modeling problems summarized by Andrej Karpathy. Diagram from Karpathy's blog [14]

.

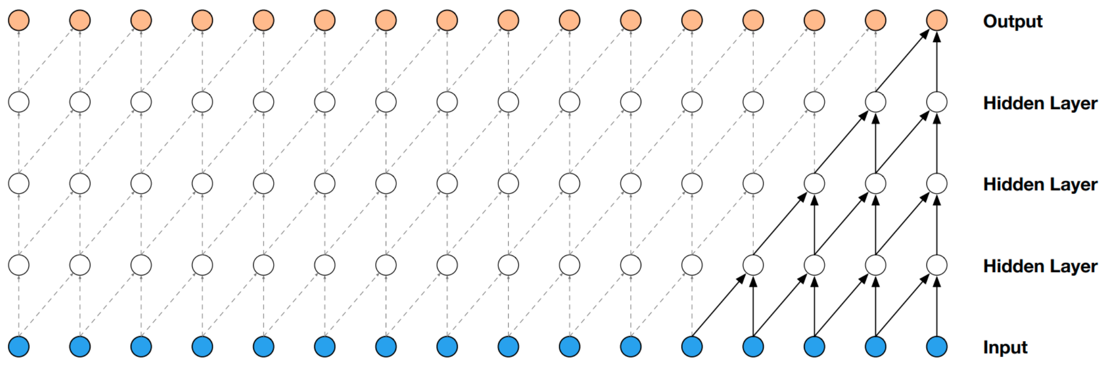


Figure 4. Diagram of a causal convolutional network for sequences.

doesn't have to travel as far backward through the network during back-propagation – there is a direct route from the error to each intermediate activation. Finally, attention is very interpretable – the network can tell you exactly which tokens or input features it looked at in order to make its decision on the prediction task. One drawback of attention is that its memory requirement is $O(L)$, where L is the number of tokens.

Attention caught hold quickly, and people even started applying attention-inspired principles in image domains. But the iteration of attention-based sequence modeling was not yet complete.

Self-attention and the Transformer

The next leap came from Ashish Vaswani, who in 2017 published "Attention is all you need" [22], advocating an entirely new sequence-based architecture. He and his co-authors called this architecture the "Transformer" neural network. The architecture bears striking resemblance to attention-based RNNs, but it has some key differences. Principally, it uses a strategy that the authors dubbed "self-attention".

The Transformer seems to solve almost every problem present in RNNs and other sequential architectures. It solves the problem of modeling long sequences without losing resolution by passing a context vector through L layers. It removes the sequential training constraint, allowing for batch training. It allows global relationships (from one side of the sequence to the other) as opposed to the local relationships modeled by WaveNet. With this architecture, Vaswani et. al. set new benchmarks for neural machine translation.

The one drawback to self-attention is that it requires $O(L^2)$ memory, where L is the length of the input sequence. However, researchers have

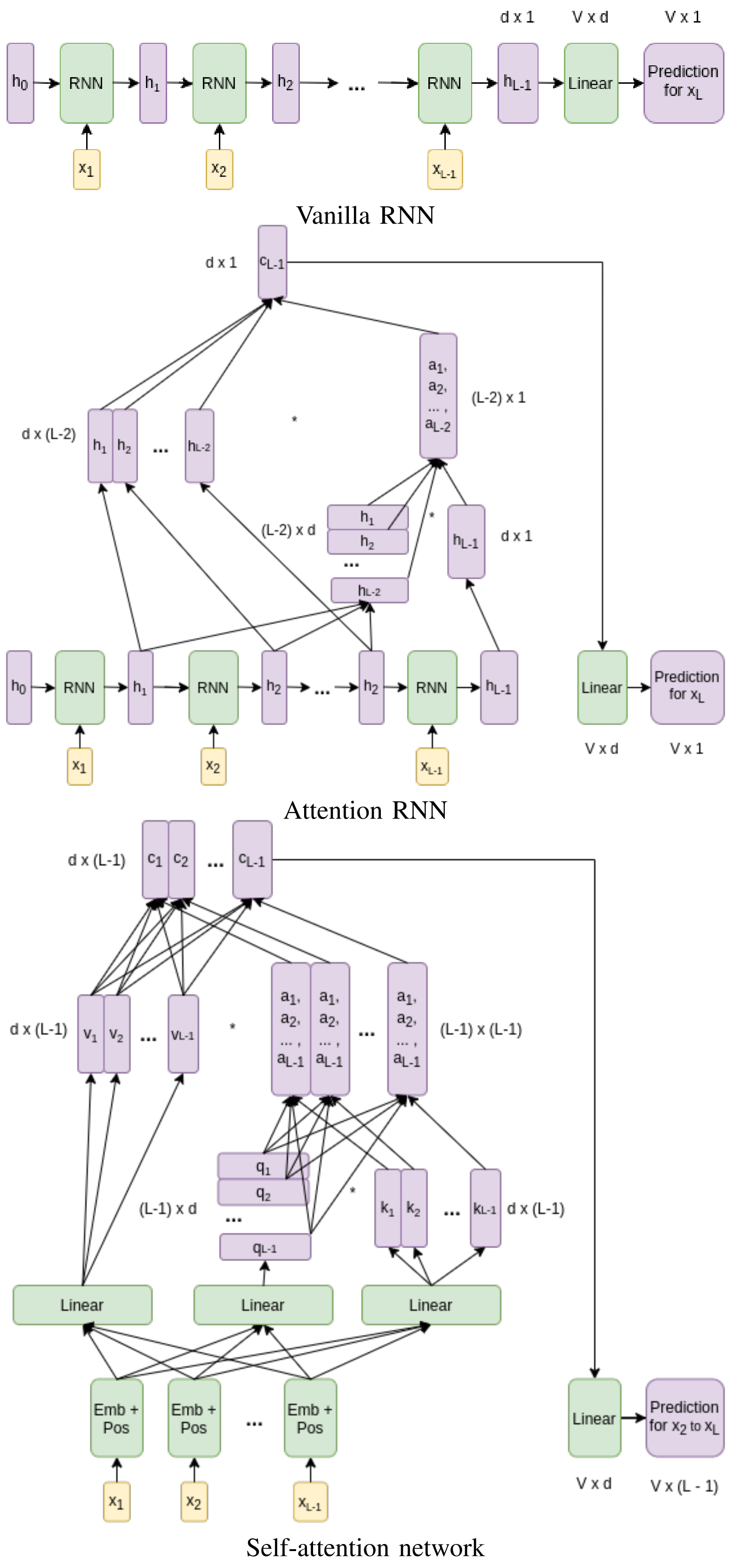


Figure 5. A comparison of a vanilla RNN, an attention-based recurrent network, and a self-attention-based network. These are not accurate to fine implementation details, but rather are diagrammed to illustrate the fundamental commonalities and differences between the architectures.

already begun to solve this problem; for example, the "Performer" achieves similar performance to the Transformer with linear memory requirements and no major compromising assumptions (i.e. sparsity) [23].

Interlude 2: Technical tutorial on Transformers

We will now examine the Transformer architecture, and the advances that it offers over traditional attention RNNs. We will first cover a high-level list of the Transformer's advances, then study the mathematical implementation of self-attention.

List of Transformer advances

While reviewing this list, the reader is encouraged to examine Figure 5 and identify these differences between the Attention RNN diagram the Self-attention diagram. 1: The Transformer computes a attention-based context vector not just for the next token, but for all of the tokens. 2: It is not sequential. Due to the fact that the network can attend to any point in the input sequence, there is no longer any reason to pass information sequentially through the network. This allows for easier batch training, eliminating a major bottleneck for RNNs. 3: It uses different linear projections for each vector involved in the dot product. It terms these projections, respectively, "queries", "keys", and "values". This allows the network to learn what to attend to, rather than each token only attending to tokens with a similar embedding. 4: It adds multiple attention "heads" (not pictured in Figure 5), allowing the network to simultaneously attend to multiple parts of the sequence, if doing so will be beneficial for learning. 5: It includes "positional encodings" which it adds to each input token embedding. This gives the network a point of reference to learn sequential relationships between the input tokens, rather than creating a linear combination context vector containing no information on the relative positions of the tokens.

Mathematical implementation of self-attention

To compute one self-attention layer, we start with an input tensor of dimension L x d, containing the embeddings for each token in the sequence. Then for each of h attention heads, we multiply that layer by three weight tensors of dimension d x $\frac{d}{h}$, which we will call W_q, W_k, and W_v, to obtain three L x $\frac{d}{h}$ tensors, which we call Q, K, and V. Having a Q, K, and V for each attention head allows the network to pay attention to different parts of the inputs simultaneously. Next, we multiply these tensors as follows:

$$A = QK^T \tag{1}$$

This gives us an L x L tensor of attention weights. We apply the softmax function across the second dimension of A, which will result in attention weights that sum to 1 across the L dimension. Finally, we apply a scaling term $\frac{1}{d/h}$ and multiply this tensor by the value tensor V to obtain an L x $\frac{d}{h}$ tensor.

$$B = \frac{softmax(A)V}{\frac{1}{d/h}} \tag{2}$$

Lastly, we concatenate the attention heads back together, leaving us with an output tensor of dimension L x d, the same size as the input.

$$C = Concat(B_i) \qquad \forall i \in \{1 \ldots h\} \tag{3}$$

GPT and beyond: pre-training and model scaling

Since the Transformer and self-attention, most recent advances in neural sequence models have come via iterations on the Transformer model.

The Transformer was originally introduced with an encoder-decoder architecture to handle machine translation. Later iterations that were based on NLP for a single language used only the encoder [24] or decoder [25] half of the Transformer. Jacob Devlin, with his BERT model, adopted the techniques of bi-LSTMs to train a transformer that could look at context tokens on both sides of the token to predict [24].

But interestingly, the greatest improvements came from two relatively unoriginal techniques: pre-training and model scaling. In 2018, Alex Radford published a paper on a "generative pre-trained transformer". He first trained a transformer decoder on a book text dataset in a self-supervised way, training it to predict the next token in the sequence. He then fine-tuned this model by adding a single trainable layer at the end of the network for each task he wanted the network to perform, then training on a smaller dataset related to the fine-tuning task. The pre-trained/fine-tuned models vastly outperformed models with the same architecture that were trained without fine-tuning. Later papers, including GPT-2 [26] and GPT-3 [27], increased the model size from millions of parameters to as many as 175 billion parameters, and saw continued increased performance with increased model size.

The recent advances in sequence modeling illustrate the shift in research that occurs when algorithms become competent for practical applications. In the past 10 years there have been a few significant algorithmic improvements to neural sequence modeling (attention and self-attention). But much of the other research has centered around simple applications of existing technologies to new data and scaling of model size to increase performance. It is worth thinking about this shift from both a research perspective and a business perspective to weigh the potential benefits and drawbacks of where research efforts are spent.

Conclusion

It will be interesting to see how sequence modeling evolves in the future. There are many applications to which transformers and self-attention have yet to be applied. Additionally, there could be improved neural network architectures. Will a better network architecture replace attention? Are there better ways to learn from sequential data than with neural networks? Neural networks are connectionist models – that is, they operate on continuous data and continuous relationships between data. The myopic focus of the 2010s on connectionist models may prevent us from seeing better sequence modeling solutions that rely on symbolic modeling principles.

Another open question is how to learn from only a few examples. For a human, a single example of a sequence is often enough for them to generalize and recognize other sequences of that class. Is it possible for a network to attain sample efficiency at a similar level? GPT-3 [27] showed some results indicating that trained language models can be used for one- or few-shot learning, but it is worth investigating whether the GPT-3 technique of pre-training accurately models the human one-shot learning experience. If not, there could be better human-inspired methods for learning to recognize sequences in a single shot.

Neural sequence modeling has grown from a proposed solution for modeling higher order functions into an application-heavy field oriented on real-world problems. Along the way, fundamental advances in sequence modeling network architectures have converged on a self-attention based network architecture with the power to learn complex global dependencies over long sequences of text. These models have been enlarged in number of parameters and trained with specialized hardware accelerators, elevating them to human-like performance and beyond in a variety of sequence modeling tasks.

■ REFERENCES

1. W. S. Mcculloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, p. 115–133, 1943.
2. F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain.,” *Psychological Review*, vol. 65, no. 6, p. 386–408, 1958.
3. M. Minsky and S. Papert, *Perceptrons*. 1969.
4. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation,” tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.
5. S. Linnainmaa, “Taylor expansion of the accumulated rounding error,” *Bit*, vol. 16, no. 2, p. 146–160, 1976.
6. S. Dreyfus, “The numerical solution of variational problems,” *Journal of Mathematical Analysis and Applications*, vol. 5, no. 1, p. 30–45, 1962.
7. H. J. Kelley, “Gradient theory of optimal flight paths,” *ARS Journal*, vol. 30, no. 10, p. 947–954, 1960.
8. C. Olah, “Understanding lstm networks,” Aug 2015.
9. S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, p. 1735–1780, 1997.
10. A. Graves and J. Schmidhuber, “Framewise phoneme classification with bidirectional lstm networks,” *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*

11. A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), vol. 25, pp. 1097–1105, Curran Associates, Inc., 2012.
12. K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” 2014.
13. I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Advances in Neural Information Processing Systems* (Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, eds.), vol. 27, pp. 3104–3112, Curran Associates, Inc., 2014.
14. A. Karpathy and L. Fei-Fei, “Deep visual-semantic alignments for generating image descriptions,” 2015.
15. A. Karpathy, “The Unreasonable Effectiveness of Recurrent Neural Networks,” May 2015.
16. E. Mansimov, E. Parisotto, J. L. Ba, and R. Salakhutdinov, “Generating images from captions with attention,” 2016.
17. N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent, “Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription,” 2012.
18. A. Graves, A. Mohamed, and G. Hinton, “Speech recognition with deep recurrent neural networks,” in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 6645–6649, 2013.
19. Z. Wu and S. King, “Investigating gated recurrent neural networks for speech synthesis,” 2016.
20. A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, “Wavenet: A generative model for raw audio,” 2016.
21. D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” 2016.
22. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” 2017.
23. K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser, D. Belanger, L. Colwell, and A. Weller, “Rethinking attention with performers,” 2020.
24. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” 2019.
25. A. Radford, “Improving language understanding by generative pre-training,” 2018.
26. A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language models are unsupervised multitask learners,” 2019.
27. T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, “Language models are few-shot learners,” 2020.

Jacob Stern is a PhD student applying neural sequence models to protein structure prediction and protein design. In his free time he enjoys playing piano and skiing with his wife, Kimberlee.

Bayesian Deep Learning for Uncertainty Estimation

Bradley Hatch
Brigham Young University

***Abstract*—Deep learning has made tremendous strides towards artificial general intelligence (AGI). These complex and hierarchical models eliminate much of the need for human engineered model inputs, and are custom designed to ingest specific types of data (e.g. images and text). Because neural networks have millions of parameters, they need large amounts of data from which to learn. Without a sizable dataset these models can quickly overfit to the training data, resulting in a model that is underspecified. To overcome this problem one can apply principles from Bayesian inference to quantify a model's uncertainty in its predictions, and thus revealing what a model ”knows”. This paper is a review of the challenges and advances in identifying epistemic uncertainty in neural networks.**

■ **POWERFUL MACHINE** learning models have revolutionized the pursuit of artificial general intelligence (AGI), a subfield of machine learning that has recently attracted much attention is deep learning. An event that led to the awakening of deep learning from its A.I. winter happened in 2012 when Alex Krizhevsky won the ImageNet image classification challenge [1] by a significant margin. Krizhevsky used a innovative convolutional neural network (CNN) [2] to model the raw pixel values of the complex images. There are several reasons why this seminal event captured the eye of the media and machine learning researches alike; one such reason was that it represented a transition of the burden of representing data to a mathematical model from human experts, limited by their own experience and imagination, to large, multi-layered models. The complexity of images was not distilled to a single vector of human engineered features. Rather, they were left as raw pixel-valued inputs, which placed the onus of feature importance on the CNN. Many advancements in deep learning have been made since 2012, but these models still remain difficult to interpret, and consequently, are often referred to as ”black-boxes”.

It is difficult to know what a neural network does and does not know, or even how it represents knowledge. Grant et al. [3] claim that a model's knowledge can be measured by how quickly it can adapt to unseen tasks, rather than an embodiment of task specific knowledge in its parameters, or weights. The former emphasizes *how* a model learns as opposed to the later, which focuses on *what* a model has learned. Both philosophies require data for a model to learn. In fact, deep learning models require large quantities of data in order to sufficiently define their millions of parameters. With insufficient data, these models can quickly overfit to data from which they learn, which leads to poor predictions of future data. This is a common occurrence in classification tasks when using deep models. A model that is underspecified by the available data will still attempt to classify future data into a predetermined category, even though its knowledge-base is found wanting. This led to the need to quantify the uncertainty in a model's predictions.

Epistemic uncertainty is the uncertainty over the set of parameters of a model. This uncertainty can be interpreted as the amount a model has learned. Quantifying this can give an indication of whether a model is well specified by the data. It can also give it the ability to abstain from classifying a data point if its uncertainty in that prediction is high. With regards to the entire dataset, high overall uncertainty signals a low general knowledge of the task. This is a challenge that is well suited for Bayesian inference to answer.

THE IMPOSITION OF KNOWLEDGE

Before exploring Bayesian approaches to deep learning, it necessary to describe model estimation from two statistical perspectives: Maximum Likelihood Estimation (MLE) and Bayesian Model Averaging (BMA), an application of Bayesian inference. Both methods are used to estimate a model's unknown parameters, but they approach this goal differently. MLE searches for the single best set of parameters given the data on hand, i.e. training data, while BMA averages all possible sets of parameters weighted by each set's likelihood. This distinction is a necessary step in order to illuminate the conceptual areas where Bayesian principles can be applied, and why they contain attributes fit for artificial general intelligence. One such concept is the ability to apply prior knowledge to decision making, and being able to update that knowledge in light of new information.

One can not fully avoid asserting human bias and judgement when modeling data. For example, feature engineering, the process of domain experts extracting features from data, is a major implicit bias placed on the data. Those features are limited by the expertise and knowledge of the human experts. Although deep learning does not suffer as greatly from this limitation, even the selection of which model to use is a form of a priori information, albeit an implied one. The following example is a high-level walkthrough of implied bias in model selection, and the conceptual differences in the two methods.

30,000 Foot Example of Method Difference

A goal of statistics is to estimate unknown parameters of a distribution, or model, using information contained in a dataset. A beginning approach one might take is to identify a distribution from a family of known distributions (e.g. Normal, Uniform, Exponential, Beta) to best model the training data D. This goal should not be conflated with the task of inference, or using an already trained model to predict new data. The learning problem can be stated as trying to find the best weights w for the function $f(x; w)$. One way to accomplish this is by finding the w that maximizes the following distribution:

$$p(w|D) \tag{1}$$

In words, the information contained in the training data D is used to help search for the most likely set of unknown parameters w. It is a distribution over all the possible combinations of parameter values. Anything to the right of the pipe ("|") is information (e.g. data, models, event occurrences) one has on hand to aid in the search. Without it, all parameter combinations of all values would have to be considered, an impossible task indeed. The choice of $p()$, or model approximating $p()$, depends on the assumptions about the data. For example, say a dataset was create by surveying 100 students and measuring their height. If the Uniform distribution is chosen to model these data, then the assumption is that all heights are equally likely to appear in the dataset. On the other hand, if the Exponential distribution is selected, then it is assumed there are exponentially more tall students than short students on campus. Of course, in the end the Normal distribution would prevail, implying that most students will be close to average height with very tall and very short students occurring less often. By selecting the Normal distribution one imposes an implicit bias to the modeling process. This implicit bias is a form of knowledge, or prior belief, that the researcher uses to select the appropriate model. Now that a model has been selected, its unknown parameters need to be estimated.

MLE

The next step is to estimate the unknown parameters of the Normal distribution, μ and σ^2. μ is the mean height of the student sample, while σ^2 is the variance of the heights in the sample.

Substituting μ and σ^2 for w in (1), the distribution is as follows:

$$p(\mu, \sigma^2|D) \tag{2}$$

This distribution covers all the possible values of μ and σ^2, given D. Some parameter values will be more likely to explain the data than others, and thus will have a higher joint probability. For example, μ (mean height of students) is less probable to be 100 feet tall than it is to be 5 feet tall. One way of solving this problem is to reverse the order of information on hand and search for the best μ and σ^2 that make the data most likely. This is called the likelihood function, and maximizing it looks like this:

$$argmax_{\mu,\sigma^2} p(D|\mu, \sigma^2) \tag{3}$$

This is Maximum Likelihood Estimation (MLE). The result of MLE is point estimates, or simply two scalar values, of the parameters. Point estimates suggests a compression of all information, including possible variations contained in D, into a single value (or values). When optimizing this equation it is common to work with the natural logarithm of the likelihood function. This is arguably the most popular loss function in deep learning, the negative log-likelihood loss (minimizing the negative of the log-likelihood is the same as maximizing the log-likelihood as in (3)), which is also an indication of a need for probabilistic approaches. Both in this simple example and in deep learning, MLE produces a single estimate of the unknown parameters. In the case of deep learning, the number of parameters can number in the tens of millions.

Questions emerge when examining the concept of point estimates. How certain are these estimates? If the MLE for the data produces μ = 5'6", would not μ = 5'7", or μ = 5'5.59203" also be good choices? These questions are related to the epistemic uncertainty regarding the parameters. Bayesian methods address epistemic uncertainty.

THE BAYESIAN APPROACH

Instead of solving for the single best choice of unknown parameters values, what if one could examine how the parameters are distributed? This shifting in perspective means the parameters are now represented as random variables instead of point estimates. The less randomness in these random variables (sharply peaked distributions) the more they look like point estimates. Philosophically, this approach starts from from a statistical inability to know which estimates are the exact answer, but assumes the distributions over the parameters contain the correct setting. Equation (1) is called the posterior probability distribution, or simply the posterior. The posterior is a distribution over all possible sets of parameters that exist given D. It is an amalgamation of all present data and information, and can be approximated using a combination of the sum and product rules of probability to produces Bayes' Theorem:

$$p(w|D) = \frac{p(D|w)p(w)}{p(D)} \tag{4}$$

Unlike the likelihood function, this is a true probability distribution. Next is a close examination of each part of Bayes Theorem, both mathematically and conceptually.

Training data D

The current available data that will define the parameters for a model. Typically, when one declares a set of data as a training set it implies that it is a good representation of all data related to an experiment. If the task is classification then D consists of (x, y) pairs where x is the input data, and y is the associated label.

Unknown Parameters w

These are the unknown values of the model. The training data will help mold and define an appropriate set to govern the model. The more training data, the better the parameters can be approximated. Well chosen parameters give the model power to make accurate inferences on future data. The parameter search space is defined by the selected model. In the example of student heights, the search space is R^2, because there are only two parameters. But for a deep learning model, the space can be as big as (or bigger than!) R^{1e10}.

Prior Distribution $p(w)$

This is where any wisdom or information about the parameter set is distilled into the process of estimation before any learning from the data has occurred. One might ask, what is known about the problem? What is known about the parameter values? If the answer is, "Nothing", then one might set the prior as the Uniform distribution. Using the Uniform distribution implies that all choices of parameter sets are equally probable. But looking at the student height problem, the average height will not be negative, and possibly not below 3'0". This can be can asserted by stating that the unknown parameter μ is distributed normally with mean = 5'5" and variance = 1'. After the estimation of the posterior happens, more data may become available. In this case, the posterior distribution, which does not contain information about the new data, becomes an excellent candidate to replace the previous prior distribution. Thus, the prior distribution needs to look a lot like the posterior distribution, i.e. have the same functional form. When this happens, the prior and posterior form a conjugate pair. If a poor choice is made for the prior it can be overcome with an increase in the amount of data. This principle is encapsulated in the classic Bernstein–von Mises theorem which states with enough data, and some not too restricting conditions, the initial choice of prior doesn't matter as long there is enough data.

Marginalization Factor $p(D)$

Marginalization is at the heart of Bayesian inference. The posterior is a conditional probability, meaning it is a probability distribution of a random variable, but only after some other event has happened. This will affect the distribution of the posterior. For example, if a couple will have two children, then the probability of both children being girls is $\frac{1}{4}$, $\{$(g,g), (g,b), (b,g), (b,b)$\}$. However, if the event of the first child's birth has occurred, then the probability of the couple having two girls is $\frac{1}{2}$, $\{$(g,g), (g,b)$\}$. The knowledge of the gender of the first child changes the probability space from four equally possible outcomes to two. Dividing by $p(D)$ integrates out, marginalizes, or normalizes the probability distribution of the original problem.

Likelihood Function $p(D|w)$

The likelihood has already been introduced. This estimates how likely the data is to occur given a certain set of parameter values. The likelihood function is *not* a probability function as its integral does not always equal 1. A low likelihood means the set of parameters is a poor choice, while a high likelihood indicates a good set of parameters.

Posterior Distribution $p(w|D)$

This is the distribution of all the possible sets of unknown parameters w. It is learned from data, and the goal of learning. If, for example, the previously mentioned μ and σ^2 of the Normal distribution are the unknown parameters, then the posterior contains knowledge of how μ and σ^2 are distributed *after* the model has seen the training data. Again, this is not a point estimate of the parameters, but a probabilistic distribution over all possible sets of values of μ and σ^2.

Predictive posterior distribution

Although the posterior is critically important, the ultimate goal is to use the posterior to predict future data. This is called the predictive posterior distribution.

$$p(y|x_*, D) = \int p(y|x_*, w)p(w|D)dw, \quad (5)$$

with x_* as the new test data not included in training the posterior. The predictive posterior distribution is used for inference on new data. Equation (5) says in order to predict the unconditional (not conditioned on any unknown parameters) label y of new data x_*, given x_* and the past training set D, integration (or summation) over all possible sets of parameters multiplied by their respective posterior probability is executed. Simply, a weighted sum of all possible models is calculated! This is known as Bayesian Model Averaging (BMA).

The goal of Bayesian learning is to obtain the predictive posterior distribution over all models, by first solving for the posterior distribution for all parameters. Interestingly, as more and more data are obtained, the posterior in turn gets more focused on a set of parameters until it finally collapses on a single set of point estimates. In other words, with enough data the posterior approximation is equal to the MLE. Upon hearing this, one might posit the question, “Are they really different methods if they both eventually collapse to the same answer?” Conceptually, yes, they are different. Here is an example to show their difference.

Good Ol’ Coin Flip

Modeling the probabilities of flips of a coin is a well studied scenario, and mentioned in nearly every entry level probability course. But it would be unwise to allow this seemingly pedestrian scenario to detract from its ability to illuminate the beauty of Bayesian reasoning. The discussion begins with the result of the first flip. Heads (**pause for applause**). This is the only data point thus far; one trial, one heads. Say the task is to model the outcome of subsequent flips of this coin. It is already evident that the model will be underspecified by the data consisting of one example. It is natural to be drawn toward the Binomial distribution to model this problem. The Binomial distribution has but one unknown parameter that defines its distribution, λ. This is how λ would be estimated from the MLE perspective:

n = number of flips
m = number of heads
$D = (y_1, y_2, \ldots, y_n)$, flip outcomes
λ = the unknown probability of getting heads

Recall, that MLE is not a probability distribution. Thus, one must use optimization methods to maximize the likelihood. The likelihood function for λ is as follows:

$$L(D) = p(D|\lambda, m) = \binom{n}{m}\lambda^n(1-\lambda)^{n-m}, \tag{6}$$

and MLE optimization is stated like this:

$$MLE(D) = argmax_\lambda p(D|m, \lambda) = \frac{m}{n} \tag{7}$$

Substituting m=1 and n=1 results in the next flip being heads with 100% probability. Does this make sense? Of course not. The Bayesian perspective of marginalization affords more insight to this problem.

Using the same likelihood in (6) the conditional probability is set up to be proportional in distribution (ignoring the marginalization factor $p(D)$)

$$p(\lambda|D) \propto p(D|\lambda)p(\lambda) \tag{8}$$

The posterior, $p(\lambda|D)$, is modeled using a Binomial distribution. The prior distribution needs to look like (i.e. have the same functional form) as the posterior, because the posterior will replace the prior before the coin is tossed again. The Beta distribution fits that bill.

$$Beta(\lambda; a, b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\lambda^{a-1}(1-\lambda)^{b-1} \tag{9}$$

The gamma functions are for normalization. The only thing left is to choose values of a and b. The uninformative, or “I don’t know”, choice for a and b are 1 and 1, respectively. Here is the resulting Prior:

$$p(\lambda) = Beta(\lambda; 1, 1) \tag{10}$$

The posterior distribution with the unknown parameter λ is proportionally equal to the Likelihood * Prior, or:

$$p(D|\lambda)Beta(\lambda; 1, 1) \tag{11}$$

It is important to point out that BMA does not egress from the world of probabilities as is the case with MLE. This makes possible the analytical computation of the moments of the posterior. To find a good value of λ the expectation (first moment) of the posterior is taken:

$$E[\lambda|D] = \frac{m+a}{n+a+b} \tag{12}$$

Even with uninformed choices of a=1 and b=1 (equivalent to the Uniform distribution), the probability of the next flip being heads is $\frac{1+1}{1+1+1} = \frac{2}{3}$.

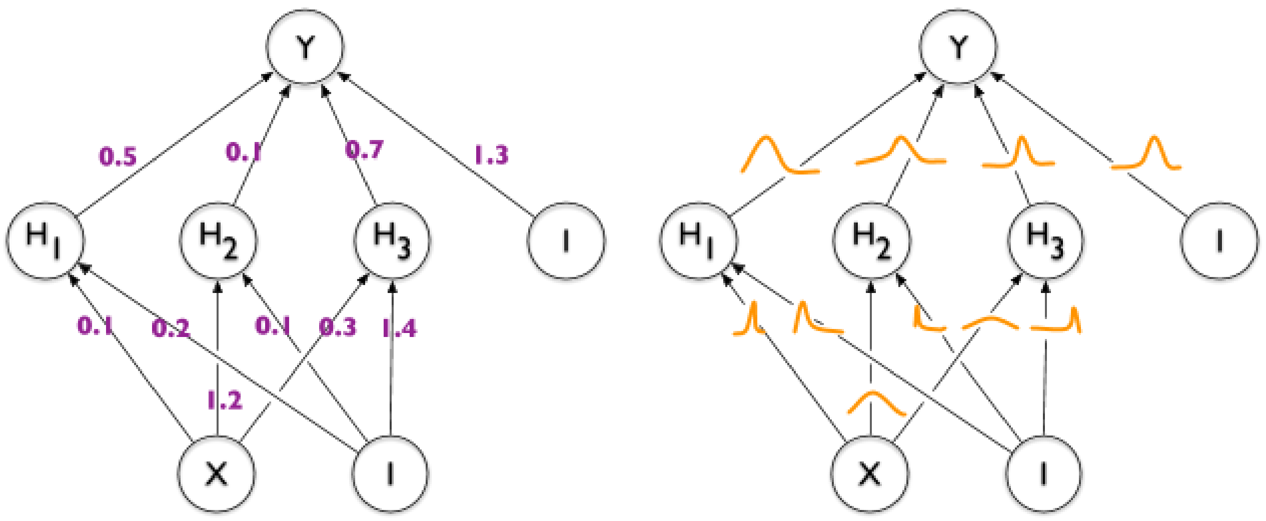


Figure 1. Deterministic neural network with point estimates on the left, and Bayesian neural network with distribution estimates on the right.)

HOW IS THIS RELATED TO EPISTEMIC UNCERTAINTY?

Thus far, the discussion has been focused on the treatment of parameters as random variables, rather than exact values to be estimated, and the conceptual differences between MLE and BMA. The solution of MLE might look like this deterministic distribution:

$$N(\mu = 3, \sigma^2 = 1.4), \tag{13}$$

where as Bayesian inference might look more like this:

$$N(\mu = N(3,1), \sigma^2 = N(1.4, 0.1)) \tag{14}$$

In (14) there is a distribution for each parameter. This means (14) is nondeterministic and will produce different predictions for the same data point. It is the variance in the predictions that determines the uncertainty in the model. The ability to calculate uncertainty is equivalent to asking the model, "What do you know?" This means that even though a data sample has a probability of 98% of belonging to class A, the model might have a low certainty in that prediction, and therefore it can abstain. Abstention in light of uncertainty is better aligned with human decision making than making deterministic decisions [4]. Using predefined distributions with only a few parameters has greatly simplified this process. What about a model with tens of millions, or even billions, of parameters? For example, the new GPT-3 language model [5] has a staggering 175 billion parameters. In the Bayesian paradigm each of the 175 billion parameters would need to have an estimated marginal distribution (see **Figure 1**). How can one apply these principles to neural networks? This highlights the need for Bayesian deep learning.

THE CASE FOR BAYESIAN DEEP LEARNING

Wilson et al. [6] claim "deep neural networks are typically very underspecified by the available data, and will thus have diffuse likelihoods p(D—parameters)." The implication of a diffuse likelihood is that there will exist a wide variety of plausible settings of parameters that can offer an explanation of the data. Millions of training samples still may not be enough to fully specify the model, which means it will be unlikely the likelihood and posterior will collapse to a single set of parameters. In the previous coin flip example, the expectation of the posterior was computed directly with the ability to define all the different parts that comprised the posterior. Unfortunately, for most models marginalization is not analytic. Markov Chain Monte Carlo (MCMC) and variational methods have been implemented to approximate (5) in the neural network setting. Neal [7]

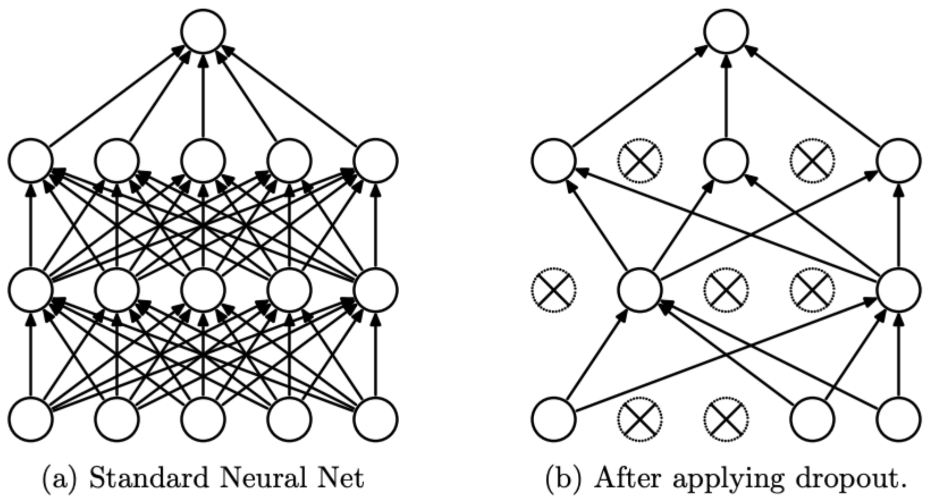


Figure 2. Adding Dropout to training randomly deletes neurons with some probability p. (a) is the standard training of a neural network. (b) shows dropout being applied to each layer.

attempted to solve the problem of integrating over the posterior with a hybrid-MCMC, a process that becomes increasingly complex with the increase in the number of parameters. Others ([8], [9], [10]) have also attempted to solve this colossal and monumental challenge, but have come up short. These attempts have led researchers to investigate other approximate methods. Khan et al. [11] use the optimizer to perturb network weights during gradient evaluations as a variation inference method to approximate uncertainty. A MCMC approach evolved by viewing Dropout, a popular way to regularizing and improving neural network generalization, through a Bayesian lens. It was cleverly coined Dropout MCMC, and is discussed in detail in the next section.

Dropout MCMC

In the early years of the deep learning resurgence, a method called Dropout was developed to prevent the multiple layers of feed-forward neural networks from becoming dependent on each other. Dropout randomly turns off, or simply multiplies by 0, the output of each node in a layer with probability p (see **Figure 2**).

Randomly deleting node outputs breaks any co-adaptation, or dependency, between adjacent layers. When training is complete, or at an intermediary inference moment to check generalization progress, Dropout is turned off and all the weights are multiplied by $1-p$. Training a neural network with Dropout dramatically increased the model's ability to generalize to new data at the cost of a slightly longer training session. The authors' motivating concept for Dropout was sexual reproduction in that it "involves taking half the genes of one parent and half of the other, adding a very small amount of random mutation, and combining them to produce an offspring". They posit a possible reason Dropout is so effective is because "over the long term, the criterion for natural selection may not be individual fitness but rather mix-ability of genes." A few years later, Yarin Gal offered a Bayesian explanation of Dropout's effectiveness.

Gal et al. [12] noticed that each parameter is dropped out with probability p, or in other words, follows a Bernoulli distribution with probability p of being 0. This is the same as "minimizing the Kullback–Leibler divergence between an approximate distribution and the posterior of a deep Gaussian process (marginalised over its finite rank covariance function parameters)." Minimizing the Kullback–Leibler divergence is a variational inference method that aims to estimate an unknown distribution (the neural network) with a simpler known distribution (Bernoulli). Viewing Dropout in this manner means the model now has a means of calculating uncertainty. For example, in classification Gal suggests obtaining an uncertainty estimate by a Monte Carlo estimate. This is accomplished at inference time by selecting a single sample and passing it through the network, but leaving Dropout active. Each time the sample passes through the model, different nodes are dropped out, thus producing a different category probability distribution. If the example is forward propagated through the network, say, 100 times producing 100 different predictions. To obtain an uncertainty measure for that single example one only needs to examine the variance of the 100 predictions. The higher the variance, the more uncertain the model is about the class prediction. The setup is different across neural network architectures (e.g. Dropout is different for CNNs than it is for RNNs), but the underlying concept is the same. This probabilistic view of Dropout makes it possible to calculate uncertainty in modern neural network architectures.

CONCLUSION

There are challenges to applying Bayesian principles to deep models: the posterior topology is difficult to navigate for models with millions of parameters, all prior information cannot be encoded as a joint distribution, and it may not be possible to view every parameter as a random variable. Despite these challenges, incredible advancements have been made that allow researchers and practitioners to investigate what their models have learned. The ability to quantify uncertainty is truly a step towards AGI.

■ REFERENCES

1. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” in *CVPR09*, 2009.
2. A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.
3. E. Grant, C. Finn, S. Levine, T. Darrell, and T. Griffiths, “Recasting gradient-based meta-learning as hierarchical bayes,” 2018.
4. L. Ziyin, Z. Wang, P. P. Liang, R. Salakhutdinov, L.-P. Morency, and M. Ueda, “Deep gamblers: Learning to abstain with portfolio theory,” 2019.
5. T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, “Language models are few-shot learners,” 2020.
6. A. G. Wilson, “The case for bayesian deep learning,” 2020.
7. R. M. Neal, *Bayesian learning for neural networks*, vol. 118. Springer Science & Business Media, 2012.
8. J. Denker and Y. LeCun, “Transforming neural-net output levels to probability distributions,” in *Advances in Neural Information Processing Systems* (R. P. Lippmann, J. Moody, and D. Touretzky, eds.), vol. 3, pp. 853–859, Morgan-Kaufmann, 1991.
9. G. E. Hinton and D. v. Camp, “Keeping neural networks simple by minimizing the description length of the weights,” 1293.
10. A. Graves, “Practical variational inference for neural networks,” in *Advances in Neural Information Processing Systems* (J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, eds.), vol. 24, pp. 2348–2356, Curran Associates, Inc., 2011.
11. M. E. Khan, D. Nielsen, V. Tangkaratt, W. Lin, Y. Gal, and A. Srivastava, “Fast and scalable bayesian deep learning by weight-perturbation in adam,” 2018.
12. Y. Gal and Z. Ghahramani, “Dropout as a bayesian approximation: Representing model uncertainty in deep learning,” 2016.

Bradley Hatch is a Ph.D. student who previously earned a B.S. degree in mathematics, and a M.S. in statistics. He is currently researching applications of information theory in neural networks, specifically identifying and amplifying fine-grain differences between cells in microscopic cytology images. Bradley is also interested in the intersection of physics, deep learning, Bayesian inference, and financial markets. In his spare time, Bradley enjoys rock climbing, snowboarding, spending time with his family, and counting by 5s.

Data Analytics in Sports

Miki Jauhiainen
Brigham Young University

***Abstract*—Data analytics is a major part of businesses and sports teams alike. One of the most important tools used in sports analytics is the machine learning method Random Forest. The Random Forest is built from many Decision Trees, which in turn utilize the concept of entropy. Entropy was first introduced by Claude Shannon in his 1948 paper A Mathematical Theory of Communication, in which Shannon establishes many important concepts and theorems, such as bits and the Shannon-Hartley theorem. In the 1960s, '70s, and '80s Decision Trees started emerging as the result of the work by Ross Quinlan, Leo Breiman, and others. Tin Kam Ho pioneered the first Random Forest in the 1990s, and it has since taken over as one of the most popular and useful machine learning algorithms. Applications of the Random Forest are tools used to monitor and improve performance, as well as tools used to predict outcomes.**

■ DATA ANALYTICS has grown to more or less control our every move. From Netflix's recommended movies to Amazon's recommended products, almost anything you see online is the result of data analytics. By analyzing the patterns and habits of their customers, companies are able to predict what movies, books, advertisements, or other goods they have to offer might generate interest as well. Data analytics is, however, not only something multi-billion-dollar companies use to make even more money [1]. It plays a pivotal role in almost every major sport, especially in team sports. Players and coaches spend hours watching their next opponent's previous match-ups to find their weaknesses in order to exploit them and gain a competitive advantage–although, as demonstrated in the 2020 World Series, it might not always go your way. While in the lead, the Tampa Bay Rays decided to pull their pitcher, who was playing very well, because of how they guessed he was going to start doing, based on their analytics [2]. The Rays ended up losing that game, and the whole series, to the Los Angeles Dodgers.

Predicting your opponents' moves is not the only way to use data analytics in sports. There are ways to analyze your own performance from a technical standpoint to help improve it. What that means is that by using tools such as computer vision and inertial measurement units (IMUs), you can analyze the arm motion of a tennis player hitting the ball [4], the arm motion of a basketball player shooting the ball [5], and the running motion of a runner [6]. Athletes can then make changes based on that data to improve their performance. The way this data transforms from meaningless values to something concrete and usable is through a machine learning algorithm, such as Random Forest.

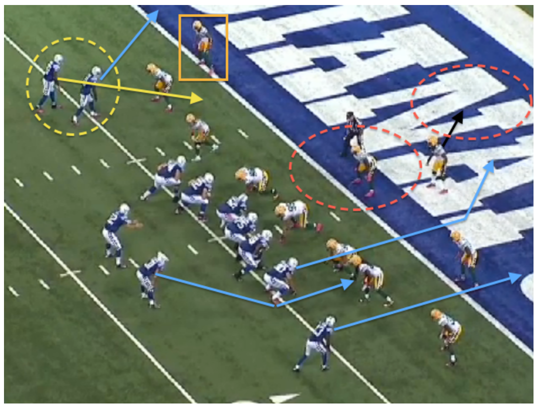


Figure 1. Analytics in football [3].

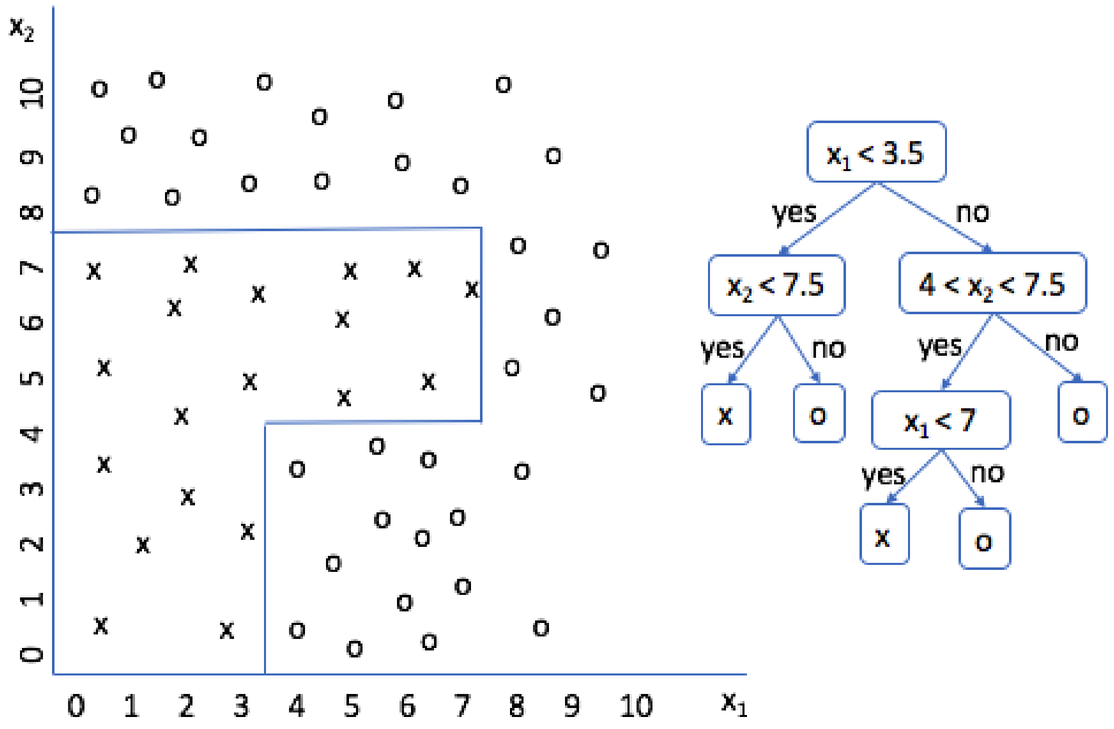


Figure 2. A simple Decision Tree and the data plot that was used to generate it [7].

Random Forest is an ensemble learning method, which means that it is a collection or grouping of other learning methods. In this case, however, there is only one other method: the Decision Tree. Basically, a Random Forest takes a user-specifiable amount of Decision Trees and combines them to get a result that is ideally better than what a single Decision Tree could achieve on its own. A Random Forest was first proposed as an improvement over single Decision Trees by Tin Kam Ho in 1995 [8], making it a fairly new algorithm. Random Forests are very easy to use, since they require little to no configuration. That, along with how accurate they are, is surely one of the reasons they are among the most popular machine learning algorithms [9].

Random Forest is a great algorithm, but it would never have come to be if it were not for the Decision Tree. **Figure 2** shows the basic functionality of a Decision Tree. The origins of the Decision Tree are unclear: multiple people published papers about tree-like algorithms around the same time, but they are all slightly different [10]. According to [10], the "first regression tree algorithm" was published by Morgan & Sonquist [11] in 1963. This Automatic Interaction Detection (AID) algorithm is a very simple binary tree algorithm that can estimate a regression function. An improvement of that, THeta AID–or THAID–was introduced by Messenger & Mandell in 1972 [12]. However, these algorithms were fairly simple and not quite robust enough for widespread use [10]. In the 1980s, three separate and more advanced classifiers were conceived. CHAID by Kass in 1980 [13], CART by Breiman in 1984 [14], and ID3 by Quinlan in 1986 [15]. All three of these are still in use to this day, which shows the significant improvement from the earlier algorithms. There is one important difference that separates ID3 from the rest, which is that it utilizes the concept of entropy.

Entropy is an important idea that comes from the field of information theory. It was first introduced by Claude Shannon in his 1948 paper *A Mathematical Theory of Communication* [16], which is one of the most important mathematical papers of the last century. Not only did Shannon come up with the formula for entropy as it pertains to information theory, but he also coined the term "bit" and laid the foundation for the whole field of information theory. Entropy, or the level of uncertainty of an outcome, is the fundamental piece of math that made everything from Decision Trees to data analytics in sports possible.

Great Minds Convene at Bell Labs

In [16], Shannon set out to extend the theories Harry Nyquist and R. V. L. Hartley had proposed in their own work a couple decades earlier [17], [18], [19]. All three men worked at the Bell Laboratories simultaneously at some point, which surely led to some fruitful exchanging of ideas. Nyquist had shown in [18] that the minimum sampling rate required to prevent loss of information is only $2B$, if B represents the bandwidth signal. Conversely, it can also be used to show that the number of pulses that a telegraph channel can handle at once is limited to $2B$. If f_p is the pulse frequency, or number of pulses per time unit, then

$$f_p \leq 2B.$$

This later became known as the Nyquist rate, and it was one of the important ideas Shannon covered in his paper. Hartley, on the other hand, implied in [19] that if the amplitude of a signal is within $[-A...A]$ volts and $\pm\Delta V$ is the precision of the receiving machine in volts, then the maximum amount of pulses is

$$M = 1 + \frac{A}{\Delta V}.$$

Then, by letting the amount of information, or number of bits, per pulse be denoted by $log_2(M)$, the line rate R–the amount of information being transferred–can be calculated using this formula:

$$R = f_p log_2(M).$$

Finally, by using the Nyquist rate, this equation could be transformed into

$$R \leq 2Blog_2(M).$$

This equation came to be known as Hartley's law, the second piece that contributed to Shannon's work in [16].

Using these findings as a basis, Shannon was able to construct the Shannon-Hartley theorem

$$C = Blog_2(1 + \frac{S}{N}),$$

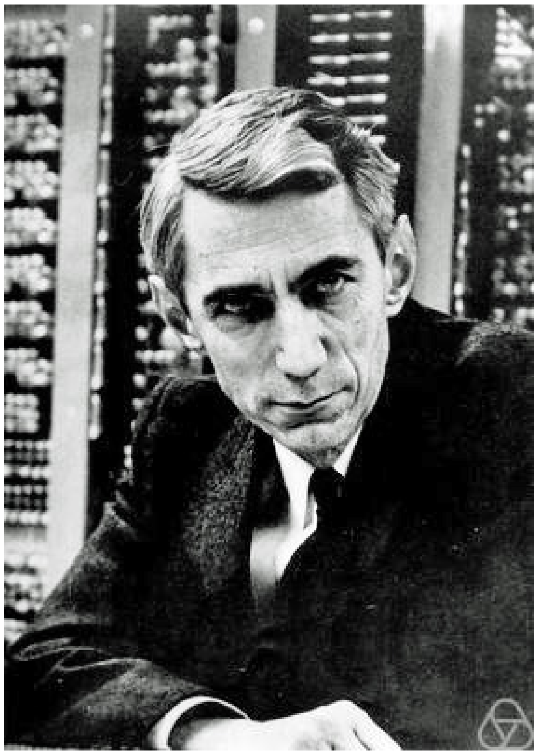


Figure 3. Claude Shannon [20].

where C is the channel capacity and $\frac{S}{N}$ is the signal-to-noise ratio. As you can see, this is almost identical to Hartley's law. Shannon was able to prove this more precise version of that law in [16]. However, even this theorem has its faults. The channel is what is called an "Additive White Gaussian Noise" channel, meaning that it is simplified to include the noise as part of the signal.

Shannon finally completes the development process of this idea when he adds entropy as part of the equation. In his noisy-channel coding theorem

$$R(p_b) = \frac{C}{1 - H_2(p_b)},$$

he states that the rate is affected by both the channel capacity and the noise. In the equation, C and R are still the channel capacity and the line rate, respectively, p_b is the probability of bit error, and H_2 is the entropy function. Shannon defines entropy in [16] as

$$H = -K \sum_{i=1}^{n} p_i log(p_i),$$

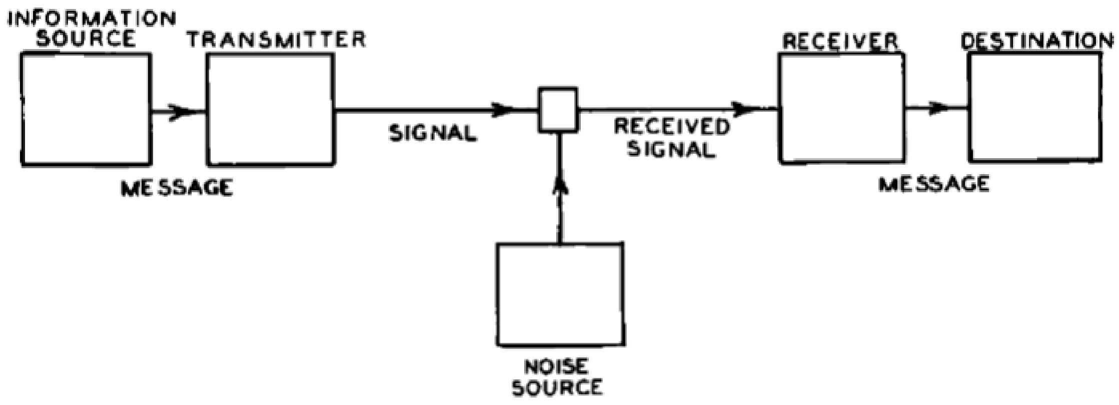


Figure 4. Diagram of a communication system from [16].

although K is merely a positive constant used to convert between units, so it can simply be 1 in most cases. The equation is taking the negative sum of the probability of each outcome multiplied by the logarithm of that probability. In the noisy-channel coding theorem, we can see that Shannon uses a special entropy function H_2. It is called the binary entropy function, and all it means is that there are only two possible outcomes (binary), and since p_b is the probability of one of them, the probability of the other one must be $(1-p_b)$. For clarity, here is H_2 written out:

$$H_2 = -plog(p) - (1-p)log(1-p).$$

From here, the field of information theory took off, and Shannon became the leading expert in it. He continued to work at Bell Labs until he joined the faculty of MIT, where he had received both his MS and PhD from, in 1956.

Information Gain in Decision Trees

Fifteen years after Shannon's groundbreaking paper, James Morgan and John Sonquist developed AID, the first algorithm resembling the modern Decision Tree [10]. It could handle both numerical and categorical values, behaving slightly differently in each case. If the variable X is numerical, the algorithm uses a comparison $X \leq c$, whereas for categorical variables the comparison is $X \in A$. A is a set of values and c is a specific value with significance for the task at hand. AID starts at a root node and splits the data at each node using one of the comparison methods above. This will result in a binary tree. The algorithm stops when the nodes reach an impurity level that is less than a predetermined amount, compared to the impurity of the root node [10]. Impurity is calculated using a deviation formula that is not relevant to this paper. THAID by Robert Messenger and Lewis Mandell allows for classification, as opposed to simply estimating a regression function like AID does. It also uses a slightly different splitting technique, but once again, it is beyond the scope of interest for our purposes. As mentioned before, these algorithms did not generate a lot of interest at the time, with AID especially receiving criticism for overfitting [10].

CHAID (CHi-squared AID), proposed by Gordon Kass in 1980, is a lot more robust than both of the earlier algorithms. It can split nodes into more than just two child nodes, and splitting is done with the help of a more sophisticated test called a Bonferroni-adjusted significance test. Another important step was made in 1984, when Leo Breiman came up with Classification And Regression Trees, or CART. It is reminiscent of AID and THAID in the sense that it uses a binary tree approach, but there are some added methods that combat the tendency to overfit. There are also ways to deal with missing values for variables. These two methods are still used today, but they are missing one piece that is important for our story: entropy.

The first person to include entropy in their Decision Trees was Ross Quinlan, who published his ID3 algorithm in 1986. The Iterative Dichotomiser 3 uses the same tree approach we are very familiar with now, but it uses entropy to decide how to split the nodes. It starts at the root node with a set S, and at each point, it calculates the entropy of each unused attribute of the set. The set is then partitioned by the element that minimizes the entropy, which, conversely, maximizes information gain. Since the idea is to gain the most information about the set, and entropy can be thought of as uncertainty, minimizing uncertainty naturally leads to maximizing information gain. Quinlan improved the ID3 algorithm even further in his C4.5 and C5.0 algorithms, making it faster and more robust.

The important feature in the ID3 and its successors is information gain. The way it works is that the entropy of the potential new state is subtracted from the entropy of the current state. This can be put as

$$IG(T|a) = H(T) - H(T|a),$$

where $T|a$ stands for the conditional new entropy value of T given the value of a, IG stands for information gain, and H stands for entropy. As mentioned above, Decision Trees in the ID3 family attempt to create the tree by maximizing information gain with each split. This works well in practice, but there are some drawbacks. For example, this method is a greedy algorithm. Greedy algorithms are known for achieving fairly good, but not always perfect, results quickly, meaning that sometimes there are better results that they miss. This applies to Decision Trees as well: because information gain is maximized at each step, the algorithm does not look ahead to see if making a suboptimal split at a certain stage results in a great split a couple steps later. By not doing so, the speed of the algorithm is greatly enhanced. However, even though Decision Trees achieved impressive results, only guaranteeing local optima left something to be desired. A certain Tin Kam Ho decided that it was not good enough.

Random Forests

In 1995, American computer scientist Tin Kam Ho published her paper on what she called "random decision forests" [8]. This was the first appearance of anything resembling a Random Forest in the literature. Surprisingly–or perhaps not–she was working at Bell Labs at the time [21]. In [8], she demonstrates how adding Decision Trees increases the accuracy of the result, as shown in **Figure 5**. As you can see, adding a new tree increases the accuracy every time, except when going from 1 tree to 2 trees. Ho attributes this phenomenon to ambiguity when combining the trees. The increase seems to be linear, although one could argue that there is some plateauing going on, depending on the algorithm. The latest version of the scikit-learn RandomForestClassifier [22] uses 100 trees by default–upgrading from 10, the previous default–so it does in fact seem like adding more trees keeps on giving better and better results. At some point the computational costs outweigh the minor improvement in accuracy.

In order to combine the different trees in the forest, Ho decided to use a linear discriminant. She credits Eugene Kleinberg with the idea–which makes sense, given that she got her PhD from State University of New York at Buffalo, where Kleinberg happened to be a math professor [21], [24]. I think it is fair to assume that they knew each other, which then led to this collaboration. The function Ho uses to combine the decision trees is called a stochastic discrimination function by Kleinberg. Here is how it works: if there are t trees, a point x is assigned to a terminal node $v_j(x)$ when descending down a tree, and the probability of x belonging to class c $(c = 1, ..., n)$ is

$$P(c|v_j(x)) = \frac{P(c, v_j(x))}{\sum_{i=1}^{n} P(c_i, v_j(x))},$$

which denotes the ratio between all class c points and all points assigned to $v_j(x)$. The actual

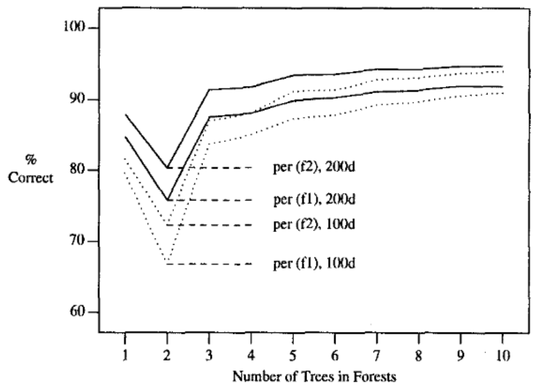


Figure 5. Algorithm accuracy per number of Decision Trees [8].

discrimination function, $g_c(x))$, is defined as follows:

$$g_c(x) = \frac{1}{t}\sum_{j=1}^{t} P(c|v_j(x)).$$

The goal is to assign x to the class c for which $g_c(x))$ is maximized. These functions are used to combine all the trees in an optimal way, which, as Ho showed in [8], leads the better results than just a single tree can achieve on its own.

How are the different trees created for use in the forest? As mentioned earlier, Decision Trees are generated using a greedy algorithm. That means that for the same input data, the output (i.e., the tree) will always be the same–the optimal output. Obviously creating the same tree n number of times will not increase the accuracy. This is where "random" comes in to play. Ho explains that the trees are created by taking random subspaces of the original data, or features. Since the random samples are all built using valid features, they will be accurate when tested against the training data. However, the differences arise when trying to generalize to data that has not been seen before. Ho also notes in [8] that there are an exponential number of subspaces to choose from, so randomization should produce completely different trees. Additionally, it means that the number of trees has to be quite large for any problems to occur.

Ho's "random decision forests" were a great foundation that Leo Breiman–the same person who came up with CART–further improved in [25]. Breiman used a method called "bagging" that he had previously published about [26] to combine the trees. Bagging, or bootstrap aggregating, is a way to generate multiple versions of a predictor, a tree in this case, and then use them to create one improved predictor. This obviously sounds very similar to the way Ho combined multiple trees to create an improved version of them: the forest. This is how Breiman describes the bagging process in [26]: given a learning set L with data $\{(y_n, x_n), n = 1, ..., N\}$, where x_n is the input and y_n is the corresponding label or output, a predictor $\varphi(x, L)$ can be formed. A learning set is equivalent to a tree in Ho's version. If we now form a sequence of k predictors $\{L_k\}$ from sets similar to L, we get a sequence of predictors $\{\varphi(x, L_k)\}$. These could be combined using an averaging approach, or something similar to Ho's stochastic discrimination; however, Breiman suggests something different. Since we rarely have multiple similar dataset we could use to create learning sets, we instead take multiple bootstrap samples $L^{(B)}$ of L, thus forming $\{\varphi(x, L^{(B)})\}$.

Taking bootstrap samples means taking random samples from L, with replacement. That means that even if a certain element or feature x_n, y_n is already in the new set, it can be chosen again.

Once we have formed $\{\varphi(x, L^{(B)})\}$, there are two possible approaches, depending on the type of data we are dealing with. If we are

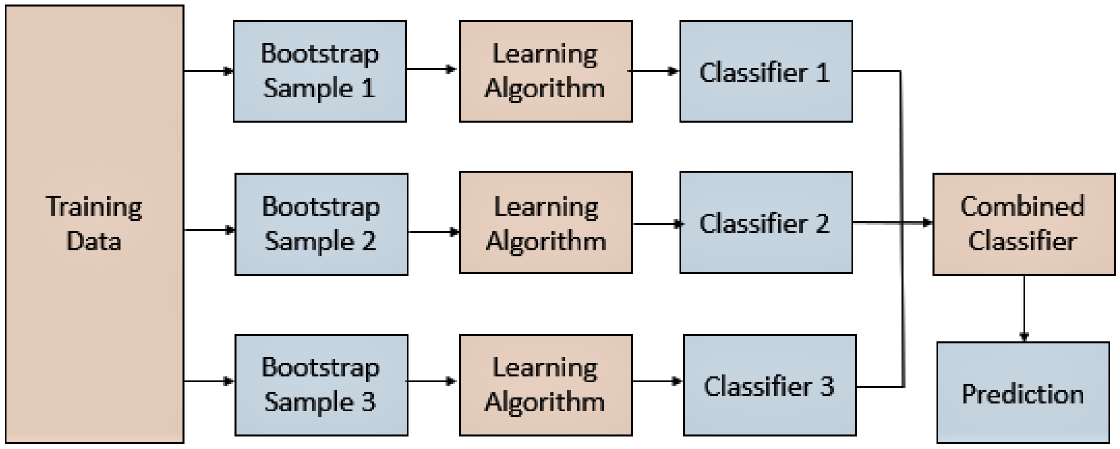


Figure 6. Forming a bagging predictor [23].

performing a regression, we can form the bagging predictor as follows:

$$\varphi_B(x) = av_B\varphi(x, L^{(B)}),$$

which is just a different way of saying that we are taking the average of the predictions. A more readable formula for this might be

$$\varphi_B(x) = \frac{1}{B}\sum_{b=1}^{B}\varphi(x, L^{(B)}).$$

On the other hand, if we are doing classification, $\varphi_B(x)$ can be formed simply by a plurality vote, meaning that of all sets $L^{(B)}$, whichever output x gets mapped to most often is deemed the correct one. **Figure 6** illustrates the bootstrapping process.

Breiman introduced an idea in [25] called the "out-of-bag estimate," which is used to estimate the generalization error. For every (y_n, x_n) in the learning set, take each L_n that does not include (y_n, x_n) in it and combine them to form an out-of-bag classifier. The error rate of that classifier on the learning set is the out-of-bag estimate for the generalization error of the bagged predictor. Out-of-bag estimates can be used during runtime to evaluate how well the classifier is going to do. They are especially useful when the learning set is small, since they can be calculated without setting aside a portion of the data to be used for validation, which means that the accuracy does not suffer from having too small of a sample size.

Another important concept introduced in [25] is the idea of variable importance. Using the out-of-bag technique described above, in addition to some additional permutations to create new tests, the accuracy of the out-of-bag classifications for each x_n can be interpreted as variable importance. This is important because in some cases there could be hundreds of features, but only a handful of them are actually important when deciding how to classify an input. **Figure 7** is the variable importance from the results of voting data, as reported in [25]. As we can see, variable 4, which correlates to the fourth topic that was voted on, is clearly significantly more important than any of the other variables. As a result, a classifier built based only on variable 4 would be almost, if not just as, accurate as one based on all the variables. Being able to reduce the number of features used for classification shortens the time needed for training and for making predictions [27]. This can make a major difference when the number of features is reduced from thousands to dozens, or even just to hundreds.

Shannon's influence can be seen in the idea of variable importance. Entropy, and information gain, which is derived from it, is all about measuring how much a certain attribute matters. The only difference between them is that entropy is used to decide how to make a split in a Decision Tree, whereas variable importance is used to decide which features to keep. In both cases, the attribute that yields the most information prevails.

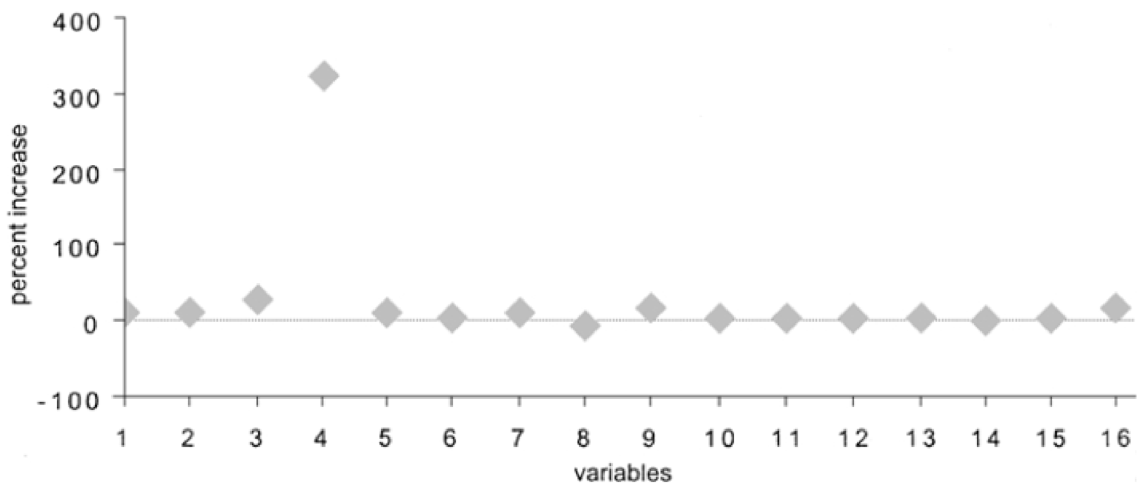


Figure 7. Variable importance in voting data [25].

After Breiman had introduced his Random Forest method using bagging, Tin Kam Ho compared the performance of their classifiers. She found that both versions are likely to perform better than a single tree, especially when the feature space is complex (it is hard to distinguish between features) [28]. In simple cases, a single Decision Tree performed as well as the ensemble forests did. This is most certainly due to the fact that adding more trees that will end up doing more or less the same thing as the single tree does not help. As far as the performance differences between the two types of forests, Breiman's bagging classifiers seemed to fair better when the data was sparse and class boundaries were nonlinear, whereas Ho's random subspace classifiers did better when the class boundaries were smoother. A class boundary is the boundary between two areas such that if an input x is on one side of the boundary it will be classified as y_1, and if it is on the other side it will be classified as y_2, for instance.

What about Quinlan's ID3 and C4.5 classifiers? Breiman naturally uses his own CART method when building Random Forests, and Ho seems to be using it as well, although she does not mention it explicitly. However, given the fact that she discusses the performance of a Random Forest built specifically from Decision Trees using C4.5 in [29], it is plausible to assume that her prior work was not utilizing them. Ho discovered in her tests that the C4.5 Random Forest utilizing her random subspace method performed better than a single C4.5 Decision Tree, a Random Forest that used bagging, and a Random Forest that used boosting. It was about 2% more accurate than any other method, regardless of how many trees were being used.

Quinlan personally seems to only have touched on the subject of Random Forests. In [30], he talks about improving the performance of C4.5 by using bagging, which essentially means that he built a Random Forest. Better accuracy is exactly what we know to expect by now.

Applications in Sports

As I mentioned earlier in the paper, machine learning algorithms, Random Forest included, are becoming increasingly popular among sports analysts. Most applications seem to fall under either improving performance or predicting outcomes.

The studies I mentioned early on are all about improving performance. IMUs typically include an accelerometer, a magnetometer, and a gyroscope. That means that an IMU can measure linear acceleration, angular velocity, and orientation. To put it simply, an IMU can always tell you if it is tilted, rotated, or moved in any direction. Smartphones have IMUs in them, that is how they know when you turn your phone sideways. Since an IMU can provide so much positional information, it is easy to track the movement of an arm, for example. Thus, in [4], for instance, the researchers were able to analyze the type of stroke (backhand, forehand, serve) that was performed. When compared to a gold standard, this could lead to being able to determine whether or not someone's technique is proper. The researchers in [5] came to the same conclusion and claim to have started work on such an application already.

The other aspect of popular Random Forest usage is predicting scores and other events. It seems like there is an increased interest in trying to predict certain events in sports, which could explain the number of mechanisms created to do so. This interest, in turn, can be explained by the growth of the sports betting industry. Ever since the ban restricting sports betting to only Nevada was lifted in 2018, over 20 billion dollars' worth of bets have been made in the U.S. alone [31]. Worldwide, the value of the sports betting market is over 85 billion dollars, and it is anticipated to grow even further [32]. Therefore, there is a lot of incentive to predict outcomes correctly. Most fans of the National Football League (NFL) have probably seen a win probability estimate on the screen while watching a game. There are a variety of different models that could be used to predict the outcome of a game, one of which is [33]. Lock and Nettleton were able to come up with a Random Forest predictor that can fairly accurately determine the win probability at each point. The mean squared error (MSE) of the predictor is 0.156, which typically might not be a great score, but when dealing with this much unpredictability, it is acceptable. The predictions naturally get better as the games go on, since there is more data available and fewer surprising things can happen with the time ticking away.

The NFL is not the only league whose games garner enough interest to warrant the creation of a win probability predictor. In [34], a Random Forest was combined with a Poisson regression to create what the authors call a ”hybrid random forest” predictor for soccer games. They used it to predict the outcome of the games in the 2018 World Cup. It was trained on the data from all the World Cups from 2002-2014, as well as other matches by the national teams from 2010 to 2018. The hybrid model had a prediction accuracy of 0.609, which, according to the authors, would have allowed for about a $70 profit when betting $100. Thus, we can see that predictors can be a useful gambling tool in many sports, especially when powered by a Random Forest.

Modern Day

Claude Shannon became one of the most influential mathematicians and engineers of the 20^{th} century. Information theory is a vital part of intelligence efforts, cryptography, cybernetics, seismic exploration, and even gambling. One of his side projects, a mechanical mouse called Theseus that was able to learn the layout of a labyrinth and exit no matter where it was placed, became one of the pioneer studies in artificial intelligence. Shannon also made highly advanced contributions that have to do with the complexity of the game of chess. Unfortunately, he later developed Alzheimer’s, which he fought for a few years before passing away in 2001.

Leo Breiman’s Random Forest became the official Random Forest when he trademarked it in 2005, shortly before his death at the age of 77. Python, the most popular language among computer scientists for machine learning purposes, has a library called scikit-learn that includes just about any tool you could imagine. As mentioned earlier in the paper, that Random Forest is based on Breiman’s CART and bagging methods. However, as demonstrated in [8], [28], [29], and [30], there are alternative solutions, if the built-in method is not ideal for your purposes. The same goes for the scikit-learn implementation of the Decision Tree.

Ross Quinlan has remained a part of the machine learning community ever since the 1980s; he is currently in charge of a company he founded in 1997 called RuleQuest [35]. RuleQuest is an Australian, like its founder, company that provides high-quality data mining tools.

Tin Kam Ho worked at Bells Labs for 22 years, focusing on statistics and artificial intelligence, as well as telecommunications–naturally [21]. She then transitioned to IBM Watson, where she remains to this day. Her work includes semantic analysis of natural languages, as well as analysis of healthcare solutions.

Thanks to the work of all of these people, as well as that of countless others, both aspiring computer scientists like myself and professional data analysts can use highly developed machine learning algorithms to perform amazing tasks that Shannon might not have even been able to imagine when he set everything in motion almost 80 years ago.

Conclusion

Decision Trees and Random Forests are vital pieces in the world of machine learning and data analytics. A major element of it is the concept of information gain, which in turn builds on Claude Shannon’s entropy. Decision Trees emerged slowly in the 1980s, after initial prototypes in the ’60s and ’70s. Decision Trees evolved into Random Forests in the mid-1990s, and both algorithms are household names among statisticians, computer scientists, data analysts, and many others. Random Forests can be used effectively when building tools to both improve and predict performances, making it a versatile and reliable component in data analytics.

Acknowledgements

I would like to thank Grant Eckstein, Claudia Mencarelli, Joseph Mak, Aatish Neupane, and Gabriel Smith for their great ideas and eye-opening feedback.

REFERENCES

1. "How netflix and amazon use data for growth (and what a/b testing can do for your business).." https://www.brillmark.com/amazon-and-netflix-personalization-and-experiments/.
2. B. Heyen, "Kevin cash's decision to pull blake snell, explained: How analytics overruled world series context clues and cost the rays.." https://www.sportingnews.com/us/mlb/news/blake-snell-kevin-cash-analytics-explained-world-series/15ja52nunza2b1b37untxyltk3.
3. "A football life." https://tcm41260.files.wordpress.com/2014/02/football-play.png.
4. D. Connaghan, P. Kelly, N. E. O'Connor, M. Gaffney, M. Walsh, and C. O'Mathuna, "Multi-sensor classification of tennis strokes," in *SENSORS, 2011 IEEE*, pp. 1437–1440, 2011.
5. B. Eggert, M. Mundt, and B. Markert, "Imu-based activity recognition of the basketball jump shot," in *38th International Society of Biomechanics in Sport Conference*, 2020.
6. A. Wixted, D. Billing, and D. James, "Validation of trunk mounted inertial sensors for analysing running biomechanics under field conditions, using synchronously collected foot contact data.," *Sports Engineering*, vol. 12, pp. 207–212, 2010.
7. J. Jordan, "Decision trees.." https://www.jeremyjordan.me/decision-trees/.
8. Tin Kam Ho, "Random decision forests," in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, vol. 1, pp. 278–282 vol.1, 1995.
9. X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg, "Top 10 algorithms in data mining.," *Knkowledge and Information Systems*, vol. 14, pp. 1–37, 2008.
10. W.-Y. Loh, "Fifty years of classification and regression trees 1," in *International Statistical Review*, 2014.
11. J. N. Morgan and J. A. Sonquist, "Problems in the analysis of survey data, and a proposal.," *Journal of the American Statistical Association*, vol. 58, no. 302, pp. 415–434, 1963.
12. R. Messenger and L. Mandell, "A modal search technique for predictive nominal scale multivariate analysis.," *Journal of the American Statistical Association*, vol. 67, no. 340, pp. 768–772, 1972.
13. G. V. Kass, "An exploratory technique for investigating large quantities of categorical data.," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 29, no. 2, pp. 119–127, 1980.
14. L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Belmont:Wadsworth, 1984.
15. J. R. Quinlan, "Induction of decision trees.," *Machine Learning*, vol. 1, pp. 81–106, 1986.
16. C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.
17. H. Nyquist, "Certain factors affecting telegraph speed," *Journal of the A.I.E.E.*, vol. 43, no. 2, pp. 124–130, 1924.
18. H. Nyquist, "Certain topics in telegraph transmission theory," *Transactions of the American Institute of Electrical Engineers*, vol. 47, no. 2, pp. 617–644, 1928.
19. R. V. L. Hartley, "Transmission of information," *The Bell System Technical Journal*, vol. 7, no. 3, pp. 535–563, 1928.
20. "Claude shannon." https://en.wikipedia.org/wiki/Claude_Shannon.
21. "Tin kam ho.." https://researcher.watson.ibm.com/researcher/view.php?person=us-tho.
22. "Randomforestclassifier." https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.
23. "Bootstrap aggregation.." https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781789136609/5/ch05lvl1sec29/bootstrap-aggregation.
24. E. Kleinberg, "Stochastic discrimination.," *Annals of Mathematics and Artificial Intelligence*, vol. 1, pp. 207–239, 1990.
25. L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 2001.
26. L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, pp. 123–140, 1996.
27. W. Koehrsen, "Improving the random forest in python part 1.." https://towardsdatascience.com/improving-random-forest-in-python-part-1-893916666cd.
28. T. K. Ho, "A data complexity analysis of comparative advantages of decision forest constructors.," *Pattern Analysis & Applications*, vol. 5, pp. 102–112, 2002.
29. Tin Kam Ho, "C4.5 decision forests," in *Proceedings. Fourteenth International Conference on Pattern Recognition (Cat. No.98EX170)*, vol. 1, pp. 545–549 vol.1, 1998.
30. J. R. Quinlan, "Bagging, boosting, and c4.5," in *AAAI/IAAI, Vol. 1*, 1996.
31. D. Purdum, "Sports betting's growth in u.s. 'extraordinary'." https://www.espn.com/chalk/story/_/id/29174799/sports-betting-growth-us-extraordinary.

32. “Sports betting market - forecasts from 2020 to 2025,” 2020. https://www.researchandmarkets.com/reports/5138739/sports-betting-market-forecasts-from-2020-to?utm_source=GNOM&utm_medium=PressRelease&utm_code=23ngfv&utm_campaign=1432653+-+Global+Sports+Betting+Market+Worth+%2485+Billion+in+2019+-+Industry+Assessment+and+Forecasts+Throughout+2020-2025&utm_exec=joca220prd.
33. D. Lock and D. Nettleton, “Using random forests to estimate win probability before each play of an nfl game,” *Journal of Quantitative Analysis in Sports*, vol. 10, no. 2, pp. 197 – 205, 01 Jun. 2014.
34. A. Groll, C. Ley, G. Schauberger, and H. V. Eetvelde, “A hybrid random forest to predict soccer matches in international tournaments,” *Journal of Quantitative Analysis in Sports*, vol. 15, no. 4, pp. 271 – 287, 01 Dec. 2019.
35. “About us....” https://www.rulequest.com/about-us.html.

Miki Jauhiainen is a first-year master’s student in Computer Science. He is studying human-computer interaction, with a focus on IMU-activities. You can also find him in the Smith Fieldhouse, where he competes for the BYU Men’s Volleyball Team.

Ego2Hands: A Dataset for Egocentric Two-hand Segmentation and Detection

Alex Lin
Brigham Young University

***Abstract*—Hand segmentation and detection in truly unconstrained RGB-based setting is important for many applications. However, existing datasets are far from sufficient both in terms of size and variety due to the infeasibility of manual annotation of large amounts of segmentation and detection data. As a result, current methods are limited by many underlying assumptions such as constrained environment, consistent skin color and lighting. In this work, we present a large-scale RGB-based egocentric hand segmentation/detection dataset that is automatically annotated and a color-invariant compositing-based data generation technique capable of creating unlimited training data with variety. For quantitative analysis, we manually annotated an evaluation set that significantly exceeds existing benchmarks in quantity, diversity and annotation accuracy. We show that our dataset and training technique can produce models that generalize to unseen environments without domain adaptation. We introduce the Convolutional Segmentation Machine (CSM) as an architecture that better balances accuracy, size and speed and provide thorough analysis on the performance of state-of-the-art models on the Ego2Hands dataset.**

■ **WITH THE RAPID GROWING** usage of wearable technologies generating massive volumes of egocentric image data [1], [2], [3], [4], the ability for machines to understand human hands becomes crucial for applications such as human-computer interaction (HCI), activity logging, gesture/sign language recognition and VR/AR since the hands play a central role in human activities and behavior. Consequently, hand detection and segmentation are fundamental in areas such as 2D, 3D hand pose estimation [5], [6], [7] and gesture recognition [8], [9]. However, hand segmentation on images in the wild is extremely challenging due to numerous factors: vastness of the color space, different skin color/texture, complex background noise, motion blur, lighting type/color, shadow features, speed and model size requirement, etc. As a result, existing color-based approaches can only perform in constrained environments with proper lighting and skin color consistent with the training data. These limitations are largely due to the lack of annotated segmentation data, a common limiting factor for segmentation tasks because manual annotation is oftentimes required but infeasible for large-scale data generation.

In this work, we aim to push the boundary for the task of real-time egocentric two-hand segmentation and detection on images in the wild (Fig. 1). Since hand segmentation and detection are highly correlated and both imperative for subsequent applications, we find it natural to tackle both tasks simultaneously.

We first address the issue of the lack of anno-

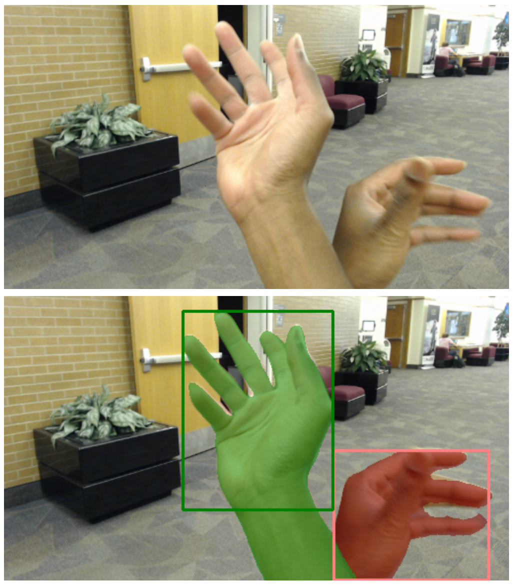


Figure 1: Our proposed dataset and training scheme enables domain generalization for two-hand segmentation and detection. Given an image with new environment and hands not present in the training data (top), the trained models can provide accurate segmentation and detection results for both hands with free interaction (bottom).

tated data. In general, real-world RGB data with segmentation ground truth is very labor-intensive to annotate. For this reason, existing hand segmentation datasets [10], [11], [8], [9], [12], [13] lack the quantity and sufficient variety necessary for learning-based approaches. Although synthetic data [7] with perfect ground truth can be generated with little cost, methods trained on synthetic image data cannot be directly applied on real-world data as Convolutional Neural Networks (CNN) are sensitive to even small textural differences between domains. We propose a novel segmentation data collection method for egocentric hands that can automatically annotate massive amount of data for only the right hand in a green screen setting, and a corresponding compositing-based data generation technique that can generate unlimited unique training instances by combination of randomly selected right hands and left hands generated using horizontal flip. In order to develop a color-invariant approach, we explore the grayscale image space coupled with edge maps as input space and show successful generalization to unseen environments. This data generation method can push segmentation models beyond the limitation of a fixed-sized training set and evaluation set and enable models to produce accurate segmentation and detection results in unseen environments without domain adaptation, which can also be easily applied to further improve model accuracy for specific environments.

We introduce Ego2Hands that includes a training set with $\sim$180,000 unique right hand instances and an evaluation set with 2,000 manually annotated frames from diverse video sequences. In-depth comparison between Ego2Hands and previous benchmark datasets shows the superiority of our dataset in quantity and diversity. For quantitative analysis, we provide comprehensive comparison between the state-of-the-art approaches on our dataset and find that existing architectures lack the proper balance of accuracy, model size and speed, which is necessary for real-world applications. To this end, we introduce a well-balanced architecture, the Convolutional Segmentation Machine, where the 1st stage outputs fast and accurate prediction and the 2nd stage provides refinement with increased resolution.

Influential Works

Convolutional Neural Networks

Similar to standard neural networks, inspired by the biological process of the animal visual cortex, the concept of using receptive fields as neurons that respond to visual features by convolution lays the foundation of modern computer vision, which is essential for color-based recognition tasks such as hand segmentation/detection.

In 1980, Kunihiko Fukushima [14] in ”neocognitron” first introduced the convolutional layer with receptive fields covering a patch of the previous layer as well as the downsampling layer. The idea later led to Yann LeCun [15] developing an approach that uses back-propagation to automatically learn the values of receptive fields from data in 1988. LeCun also introduced a convolutional network named LeNet that is capable of classifying 32×32 digits. However, limited by data scarcity and computational power, CNNs were not able to scale to more complicated visual tasks at the time.

With the development of graphics processing units (GPUs) in the 2000s, training and evaluation of standard neural networks become significantly accelerated. In 2006, Chellapilla et al. [16] introduced the first CNN with GPU-implementation for document processing.

Image-based data was also becoming more abundant as large-scale datasets such as ImageNet [17] were introduced, solving one of the biggest issue for deep convolutional networks that require large amount of training data. In 2012, as a CNN-based architecture AlexNet [18] significantly outperformed previous methods on the benchmark dataset of ImageNet, deep convolutional neural networks became the state-of-the-art approach for vision-based tasks. In 2016, He et al. [19] introduced a variant of CNN that contains residual layers with skip connections that significantly resolves the issue of vanishing gradient for deep neural networks, allowing network with over 1,000 layers to still converge. The proposed Resnet architectures achieved top results on ImageNet [17] for image classification as well as COCO [20] for object segmentation and detection. Our proposed architecture also utilizes residual layers for convolution and deconvolution.

Hand Segmentation

Depth-based methods Early works [21], [22], [23] utilized Randomized Decision Forests (RDF) on depth images to obtain the hand segmentation, which allows multicore parallelization with fast inference time suitable for real-time applications. [5] later introduced a Fully Convolutional Network (FCN) that segments the left and right hand for fast tracking of two interacting hands in egocentric viewpoint. Similarly, [24] proposed a hybrid encoder-decoder architecture with skip-connections for two-hand segmentation in third-person viewpoint. Recently, [25] extended the segmentation task to 8 classes to include arms and objects and trained a FCN on synthetic data with certain level of generalization on real depth data. Following [5], [6] used a Correspondence Regression Network to estimate two-hand segmentation prior to hand pose estimation, which shows the significance of separate segmentation of the two hands for pose estimation as it provides information on how interacting hands occlude each other. Note that for depth-based approaches, segmentation ground truth is obtained by color thresholding, requiring subjects to wear thin colored gloves. Therefore, datasets for depth-based hand segmentation are not suitable for training RGB-based approaches.

Color-based methods Depth cameras have additional setup overhead and indoor requirement with higher power consumption and cost, making its applicable applications much more limited comparing to the ubiquitous RGB cameras. Before the revolution of deep convolutional networks in the field of computer vision, [26], [27], [28] proposed motion-based approaches for binary foreground segmentation with the assumption that the motion pattern is different for the foreground and background. Some methods [29], [30], [31] rely on consistent skin color for hand segmentation. Being aware of the possible illumination difference in scenes, [11], [32] trained multiple hand detectors on a mixture of local and global appearance features from various scenes and adaptively selected detectors based on the test images. To address two-hand segmentation with possible slight inter-hand occlusion, [33] performed binary hand segmentation and left-right hand split based on the distribution of angle/position of hands as well as temporal super-pixels.

Recent approaches utilize convolutional deep networks as stronger appearance models. [8] used a CNN to classify proposed bounding boxes and performed hand segmentation using Grabcut inside the bounding boxes. They also demonstrated simple static gesture recognition using the obtained segmentation masks. Although the CNN is designed to classify detected hands as one of four interacting hands, the window proposal and classification algorithm do not address the issue of similar-object occlusion between hands. [9] later proposed to segment the hands directly using RefineNet [34] and performed extensive evaluation on multiple datasets for binary hand segmentation in less constrained environments. In order to better generalize the models to unseen scenes without requiring annotated data for training in the new domain, [13] proposed a Bayesian CNN-based approach to estimate pseudo-labels in the target domain with a hand shape discriminator for unsupervised domain adaptation. Despite achieving promising cross-dataset accuracy, their domain adaptation technique is valid for binary-label segmentation only. [7] introduced the first color-based two-hand segmentation method robust for complex interactions using an encoder-decoder residual network that also estimates the hand heatmap energy for detection. However, their model was only able to train on synthetic data and cannot be applied to images in the real-world domain. [35] trained a UNet [36] on a mixture of synthetic and noisy real-world data for two-hand segmentation in a laboratory environment from third-person viewpoint. To perform both hand segmentation and detection, we adopt the method of [7] that adds the hand heatmap energy channels to the segmentation output channels for the segmentation models in our studies. Our experiments show that the addition of hand heatmap energy output channel complements the segmentation task and does not negatively impact segmentation accuracy.

Hand Segmentation Datasets

Existing Datasets

Pioneering work [11] contributed three egocentric videos (EDSH1, EDSH2 and EDSH-kitchen) with varying illumination for training and evaluation of binary hand segmentation. For activity recognition, [10] proposed Georgia Tech Egocentric Activity Dataset (GTEA) with 663 annotated frames consisting of two-hand labels (no inter-hand occlusion). [12] later published an extended version EGTEA with 13,847 binary-label annotated frames. To enable hand segmentation in more unconstrained settings, [8] introduced EgoHands as the first large-scale hand segmentation dataset with 4,800 annotated frames consisting of a maximum of 4 interacting hands. For the same purpose, [9] additionally introduced EgoYouTubeHands (EYTH) with $\sim$1290 annotated frames from three Youtube videos and HandOverFace (HoF) with 300 annotated frames from third-person Web images. To demonstrate cross-dataset adaptation performance, [13] annotated 855 and 488 frames for human grasping datasets UTG [37] and YHG [38] respectively. To address the issue of data scarcity, [7] introduced two large-scale synthetic dataset (Ego3DHands) with a total of over 100,000 annotated frames on two hands.

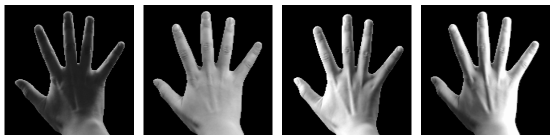


Figure 2: Hand images (grayscale) with different visual shadow features from different lighting directions.

Ego2Hands

As existing datasets with real-world data require manual annotation, they severely lack the quantity and variety needed for learning-based hand segmentation on images in the wild for real-world applications. On the other hand, synthetic dataset consists of data in a different statistical distribution from the real-world data and therefore can only be used for theoretical research analysis or mixed training with real-world data for limited knowledge transferral.

To solve the problem of data scarcity, we introduce a large-scale dataset Ego2Hands that consists of 188,362 annotated frames for only the right hand. Segmentation masks are obtained by automatically removing the background in green screen setting. 22 participants with diverse skin colors and hand features are selected and instructed to perform free one-hand motion while recording using a head-mount webcam (Logitech C922) at 30 fps. This process allows simple and fast data collection for segmentation data. During training, we composite images online by randomly selecting two right hand images, flipping one horizontally to create the left hand, and inserting a random background image. For background images, we use the 19,216 images provided by [7] with the additional 14,997 high-quality images in the DAVIS datasets[39], [40], which results in approximately 1.21×10^{15} unique hand-scene combinations prior to data augmentation.

Despite the massive quantity in training instances, it is still unrealistic for deep networks to learn the complete RGB space. For instance, for hands under a particular colored lighting, learning-based models would need sufficient training data with hands in that specific color. This is an important issue rarely addressed by previous works as their proposed datasets do not even contain diversity in skin color under normal lighting. Consequently, we explore the grayscale image space coupled with image edge maps as inputs for a truly color-invariant approach. In the grayscale domain, we find two major factors crucial for generalization in the real-world domain: brightness and shadow features. For diversity in brightness, we scale the pixel values of both hands to shift the means to a randomly selected value $\beta \in [15, 240]$ while keeping the image values always clipped within [0, 255]. Evidently, variation in the brightness of the hands also contributes significantly to diversity in skin colors. For different shadow features, we include light sources from various directions during data collection (Fig. 2 shows the visual difference in shadow features due to the direction of the light source).

To obtain the hand energy for detection, we follow [7] by generating an internal synthetic dataset (Ego3DHands$_R$) with only the right hand, and jointly train their proposed HandSegNet on Ego3DHands$_R$ and Ego2Hands using the following loss,

$$\mathcal{L}_{combined} = \mathcal{L}_{seg}^{synth} + \mathcal{L}_{seg}^{real} + \mathcal{L}_{energy}^{synth} \quad (1)$$

where $\mathcal{L}_{seg}^{synth}$ and $\mathcal{L}_{seg}^{real}$ are the Cross Entropy Loss for the segmentation of the synthetic and real right hands, and $\mathcal{L}_{energy}^{synth}$ is the Mean Squared Error loss for the heatmap energy of the synthetic right hands obtained using the ground truth 2D joint locations. We exploit the feature that both domains contain right hands with no background noise. In doing so, we successfully transfer the knowledge of the hand energy from synthetic data to real-world data and generate the hand energy for all training instances in Ego2Hands. This novel knowledge transferral method that automatically generates the heatmap energy data in the target domain is very efficient as ground truth data for object detection commonly requires extensive manual annotation [20], [41].

Some datasets [8], [9] contain annotated hand segmentation that excludes the arm, therefore combining the task of segmentation and detection into one, which is valid in absence of occlusion. Others [10], [11], [12], [13] include the arm in segmentation and neglect the task of hand detection. We argue that it is more natural to keep the two tasks separate by including the arm for segmentation because the boundary line is ambiguous. In addition, we can address hand detection in form of heatmap energy. Note that removing the arm in hand segmentation also requires manual annotation infeasible for large-scale data generation. We show in later section that our training data enables models to achieve high accuracy in both tasks simultaneously.

With the obtained hand energy, we are able to composite more realistic training images by selecting proper overlaying order. After random selection of the left and right hand from Ego2Hands, the hand with the larger energy sum is selected to be overlaid on top of the other hand. We discover that naive overlaying creates the unrealistic feature of green color bleeding at the hand boundaries, which leads to noticeable decline in the model's ability to generalize to real-world data in our experiments. Accordingly, we apply dilation and gaussian blur on the original alpha-channel to create smoother hand boundaries for overlay. The green color bleeding also becomes unrecognizable with smooth-edged overlaying in the grayscale domain.

For each composited image, we further data augment by applying 1) random horizontal and downward vertical translation within reasonable ranges on each hand, 2) random smoothing with various kernel sizes to simulate blur from motion or auto-focus 3) random brightness on the hands and background images, 4) Random horizontal flips and cropping on background images and 5) 10% drop rate for a each hand (mutually exclusive). Thus our compositing-based approach can generate unlimited training images with variety. For domain adaptation on specific environments, we simply use the background images collected from that scene for compositing training images. See Fig. 3 for generated sample images for training.

To support quantitative evaluation, we introduce an evaluation set that includes 8 videos each with 250 annotated frames. We select 4 additional participants with diverse skin tones to perform free two-hand motion in 8 different scenes with various lighting conditions. Inspired by video object segmentation method [42], we exploit the temporal consistency in video sequences by obtaining manual annotation with the

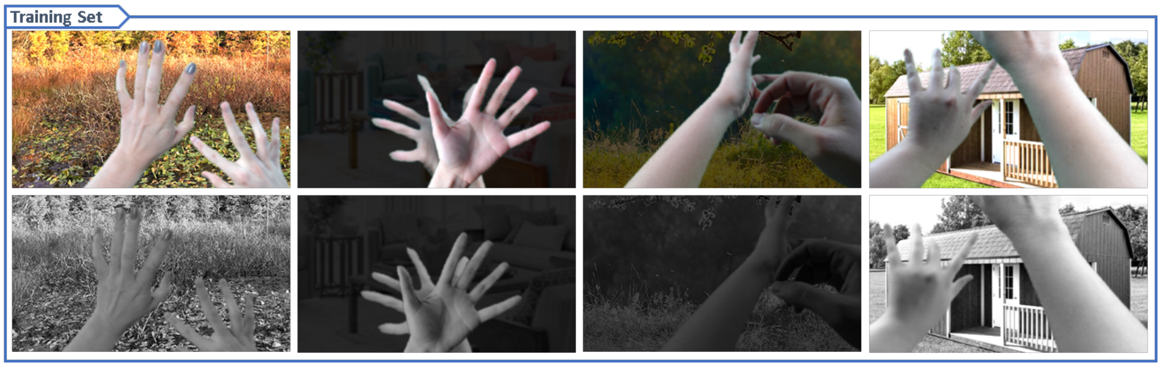


Figure 3: Top row shows the original composited images using the training set of Ego2Hands and the avaialble background images. Bottom row shows the data augmented version in grayscale for training.

Datasets	Type	#Annotated Frames	#Hand Instances	#Subjects	#Scenes	Objects	#Classes	Resolution
GTEA [10]	Real	663	1231	4	1	✓	3	720 × 405
EDSH [11]	Real	743	-	1	3	✓	2	1280 × 720
EgoHands [8]	Real	4800	15053	4	3	✓	5	1280 × 720
EYTH [9]	Real	1290	2600	-	-	✓	2	384 × 216
HoF [9]	Real	300	507	-	-	✗	3	384 × 216
EGTEA [12]	Real	13847	-	32	1	✓	2	960 × 720
UTG [13]	Real	855	-	5	2	✓	2	480 × 360
YHG [13]	Real	488	-	4	-	✓	2	480 × 360
Ego3DHands [7]	Synth	110,000	∼214,500	1	-	✗	3	960 × 540
Ego2Hands (Ours)	Real	188,362 (train) 2,000 (test)	∞ (train) 4,000 (test)	22 (train) 4 (test)	- (train) 8 (test)	✗	3	800 × 448

Table 1: Statistics of available hand segmentation datasets.

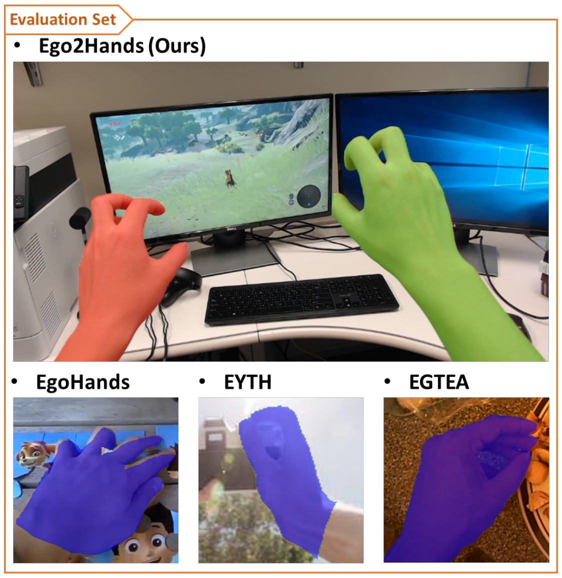


Figure 4: Illustration of the difference in annotation accuracy between datasets. Existing datasets contain false positive labeling for gaps and holes and potentially inaccurate boundaries (**Best viewed in magnification. Annotated masks are overlaid in colors.**)

help of HandSegNet[7] pretrained on Ego2Hands to minimize annotation time. For frame $\boldsymbol{F}_i$, our pretrained model produces a semi-accurate segmentation, which we manually refine with the support of Grabcut to create the ground truth $\boldsymbol{S}_i$. The model is then finetuned on $\boldsymbol{S}_i$ for a more accurate prediction on $\boldsymbol{F}_{i+1}$. We also annotate the hand energy $\boldsymbol{E}_i$ for approximate location of both hands. Fig. 4 shows a comparison of annotation accuracy between Ego2Hands and other datasets. Due to the gap between the gesture space of Ego3DHands$_R$ and Ego2Hands, some of the generated energy data also needs refinement. We use a similar tool to refine the generated energy for the entire training set of Ego2Hands semi-automatically with human supervision.

We show in Table. 1 a detailed comparison between Ego2Hands and existing benchmark datasets. Ego2Hands consists of order of magnitudes more annotated hand frames capable of generating unlimited training data with pixel-accurate segmentation annotation on both hands using our compositing-based approach. We also provide the largest and most diverse evaluation set necessary for comprehensive evaluation in various real-world practical settings. In summary, Ego2hands is superior in annotation quality, quantity and data diversity.

Convolutional Segmentation Machine

Inspired by the Convolutional Pose Machine [43] that sequentially improves the heatmap estimation for 2D keypoints in six stages, we introduce the Convolutional Segmentation Machine (CSM) that outputs segmentation prediction in two stages. The task of 2D keypoint estimation is fundamentally different from semantic segmentation as the former estimates the approximate location of the target keypoints in form of heatmap energy and does not require high-resolution output. On the other hand, semantic segmentation requires higher precision and resolution for classification of each pixel in the image.

Accordingly, we propose an encoder-decoder architecture for the task of hand segmentation as shown in Fig. 5. We couple the grayscale image and the edge map obtained from the original RGB image for a 2-channel input image and resize the image size to 288×512. Each downsampling and upsampling residual layer consists of 3 bottleneck and 3 deconvolutional bottleneck blocks respectively. For stage 1, the network output has 1/4 of the original resolution and is concatenated to the intermediate network feature. Stage 2 refines the results from stage 1 with increasing resolution that is half of the original resolution. Empirically we find that an additional stage 3 that returns the output to the original resolution does not produce higher segmentation accuracy. At test time, our 2-stage design gives users the flexibility of choosing between speed and accuracy, which is a valuable feature in real-world applications. A later section shows that both stages are capable of producing competitive results.

The segmentation output is trained on the 3 classes (Left hand, right hand and background) with cross-entropy loss. To perform hand detection as well as segmentation, we add 3 additional output channels with sigmoid activation trained using mean squared error loss for the energy regression of the 3 classes. We show in quantitative analysis that segmentation models can perform both tasks well without compromising accuracy.

Experiments

We evaluate existing state-of-the-art methods on the proposed evaluation set of Ego2Hands (8 sequences each with 250 annotated frames) and compare the two-hand segmentation and detection accuracy as well as the corresponding model sizes and inference speed. We use the mean Intersection over Union (mIoU) as the metric for segmentation task. For hand detection, we use the conventional metric of Average Precision that classifies a detection bounding box as correct if its overlap between the ground truth bounding box exceeds 50% ($AP_{0.5}$). The ground truth bounding boxes are obtained using the annotated hand energy heatmaps.

We compare the models' performance with and without the input edge map and the additional output energy channel to justify our design choices. As it is impossible for static pretrained models to produce highly accurate results in all possible scenes, we also perform experiments to study the impact of domain adaptation. To support scene-specific adaptation, we include in the evaluation set a collected background sequence ($\sim$30 seconds) for each evaluation sequence. The background collection process simulates a environment scanning procedure using prospective egocentric color-based hand tracking devices.

The following architectures are selected for evaluation:

- UNet and UNet$_{1/4}$ [36]. We evaluate using the standard UNet and a version with 1/4 of the original network width. Previous work [44] has shown that reducing the number of feature channels results in much more compact model while preserving its ability for binary-label hand segmentation.
- RecUnet and DRU-Resnet50 [44]. It is proposed that integrating recursions on the internal stages of the network can produce higher segmentation accuracy. We select RecUNet-DRU(4) and DRU-Resnet50 with Dual-gated Recurrent Unit (DRU) for evaluation as these two models achieved state-of-the-art results on multiple datasets for binary-label hand segmentation.
- SegNet [45]. Primarily motivated by scene understanding applications, SegNet is a popular semantic segmentation architecture with

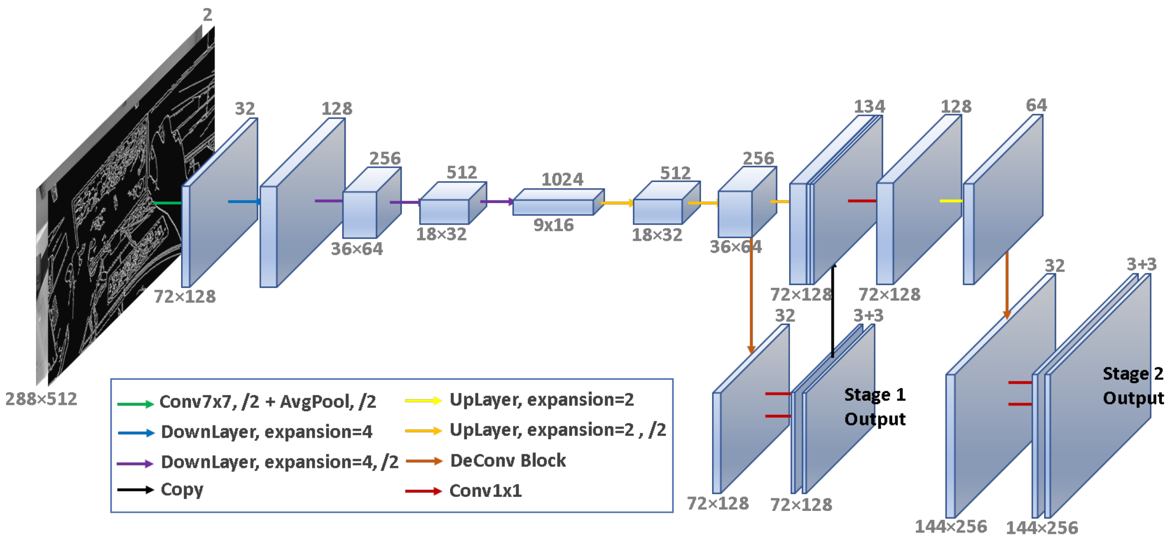


Figure 5: Architecture of Convolutional Segmentation Machine. The number of channels is denoted at the top of the boxes.

Model	#Params	Inference time (ms)	Pretrained				Adapted	
			edge ✗ energy ✗	edge ✓ energy ✗	w/ edge & energy IoU	w/ edge & energy $AP_{0.5}$	w/ edge & energy IoU	w/ edge & energy $AP_{0.5}$
UNet$_{1/4}$ [36]	0.8M	10.2	0.725	0.730	0.728	0.721	0.843	0.809
UNet [36]	13.4M	9.9	0.652	0.630	0.651	0.650	0.779	0.755
RecUNet [44]	1.1M	78.1	0.812	0.834	0.844	0.809	0.874	0.848
CSM-stage1 (Ours)	9.7M	25.4	0.721	0.802	0.792	0.827	0.878	0.910
CSM-stage2 (Ours)	10.0M	35.9	0.728	0.811	0.803	0.826	0.889	0.913
SegNet [45]	29.4M	11.9	0.687	0.667	0.644	0.637	0.788	0.776
ICNet [46]	28.3M	42.6	0.828	0.824	0.823	0.874	0.885	0.943
DeepLabV3+* [47]	59.3M	44.2	0.741	0.776	0.765	0.794	0.863	0.892
RefineNet* [34]	113.9M	49.1	0.824	0.847	0.836	0.873	0.884	0.901
DRU-Resnet* [44]	145.5M	85.9	0.001	0.275	0.306	0.311	0.550	0.507

Table 2: Evaluation of state-of-the-art models on Ego2Hands. Only IoU is reported for experiments without the energy output channel. Models with * are trained using pretrained Resnet encoder.

balance in model size and accuracy that fits well with the task of two-hand segmentation on images in the wild.

- ICNet [46]. Proposed for real-time semantic segmentation, ICNet with multi-resolution image cascade achieves high accuracy and impressive generalization with 1/4 of the output resolution. It is apparent that smaller output resolution can avoid unnecessary deconvolution layers and result in faster inference speed.
- DeepLab V3+ [47]. Targeting high-quality semantic segmentation, DeepLab v3+ improves its predecessor by adding a decoder module to further refine segmentation results. We use Resnet-101 as the encoder for this model in our experiments.
- RefineNet [34]. Using Resnet-101 as encoder, RefineNet uses multi-path refinement to exploit features available in the down-sampling process for high-quality segmentation. [9] adopted it for the task of binary-label hand segmentation.

Training Details.

We divide the training process into the pretraining and the adaptation phase. In pretraining phase, all models are trained for 100k iterations with an initial learning rate of 1.0×10^{-4} decreased with a ratio of 0.5 every 20k iterations. In adaptation phase, we train the pretrained (w/ input edge map & output energy channel) models for 10k iterations using an initial learning rate of 1.0×10^{-5} decaying with the same ratio every 5k iterations. We use Adam optimizer and find general convergence from all models using this training setup. To ensure accurate estimation, averaged results from 3 trained model instances are reported for every architecture.

As illumination (not skin tone) is the dominant factor for the brightness of the hands in input images and is known for specific environments, we perform brightness augmentation within ranges specific to the scenes in the adaptation phase. The mean brightness value β of the composited hands is scaled to be in range of [0, 55], [55,200], [55, 255] for scenes with dark (seq5), normal (seq1, 3, 4, 6, 7) and bright (seq2, 8) illumination respectively. Bright scenes has wider brightness range due to the possibility of shadow. β for background images is jittered by ± 50.

Quantitative Analysis

Table. 2 provides detailed quantitative results for the selected models on various settings. We want to point out that a comprehensive comparison involves various factors including model size, inference speed, segmentation/detection accuracy, and the ability to generalize and adapt.

Firstly, we want to justify our design choice of the additional input edge map and energy output. Models with input edge maps show overall improvement with the exceptions of UNet, SegNet and ICNet. UNet and SegNet underperform in general on the target dataset with lower generalization and adaptation accuracy. On the other hand, ICNet can achieve high segmentation accuracy without the image edge map. The addition of energy output shows minimum impact on the segmentation accuracy while providing hand detection output information essential for many applications.

With small amount of parameters, $\text{UNet}_{1/4}$ and RecUNet achieve high pretrained accuracy. However, they have worse adaptation accuracy and detection accuracy. Additionally, the recursion on internal network states notably increases the inference time, making RecUnet and DRU-Resnet the slowest models in our analysis. It is worth mentioning that inference speed for segmentation and detection is crucial in real-world applications as there are oftentimes subsequent pose estimation/gesture recognition modules. Heavy models (DeepLabV3+, RefineNet) generally achieve high pretrained accuracy as well as adapted accuracy in both segmentation and detection. We find that heavy models are dependent on pretrained encoders for optimal performance. Interestingly, DRU-Resnet with the largest model size has the lowest test accuracy despite high traiing accuracy.

We argue that it is necessary for an architecture to be well-balanced with high generalization/adaptation accuracy, compact model size and fast inference speed for practical applications. Existing models struggle to satisfy all the aforementioned qualities. To fill in the gap, CSM achieves high adaptation accuracy of IoU = 0.889 in segmentation and $\text{AP}_{0.5}$ = 0.913 in detection with only 10.0M parameters. At the same time, CSM-stage1 achieves an inference time of 0.0254s with slightly lower accuracy comparing to CSM-stage2 results. This indicates that CSM outperforms state-of-the-art models in applications with confined environments natural for scene-specific domain adaptation (VR/AR, gesture control systems, etc.)

It is important to recognize the trade-off between architectures from our experiments. For instance, in applications where scene-specific adaptation is unrealistic and generalization accuracy is required (such as egocentric sign language recognition in unconfined environments), ICNet produces promising pretrained accuracy with relatively compact model size and fast inference speed. ICNet can also be favored in applications that focus more on hand detection. In cases where the model size is not the limiting factor, RefineNet achieves very high pretrained accuracy. On the other hand, RecUnet is clearly the best model in memory-constrained settings. For applications that desire faster inference speed, shallow models such as UNet become more ideal. We claim that CSM is the most well-balanced architecture and provide insights valuable for a more thorough understanding.

We want to reemphasize that the evaluation sequences cover various range of illumination and include hands (various skin tones) and scenes not present in the training set of Ego2Hands. Our quantitative results show that the proposed dataset and compositing-based training method enables models to generalize to the real-world image domain. To provide a proper perspective for

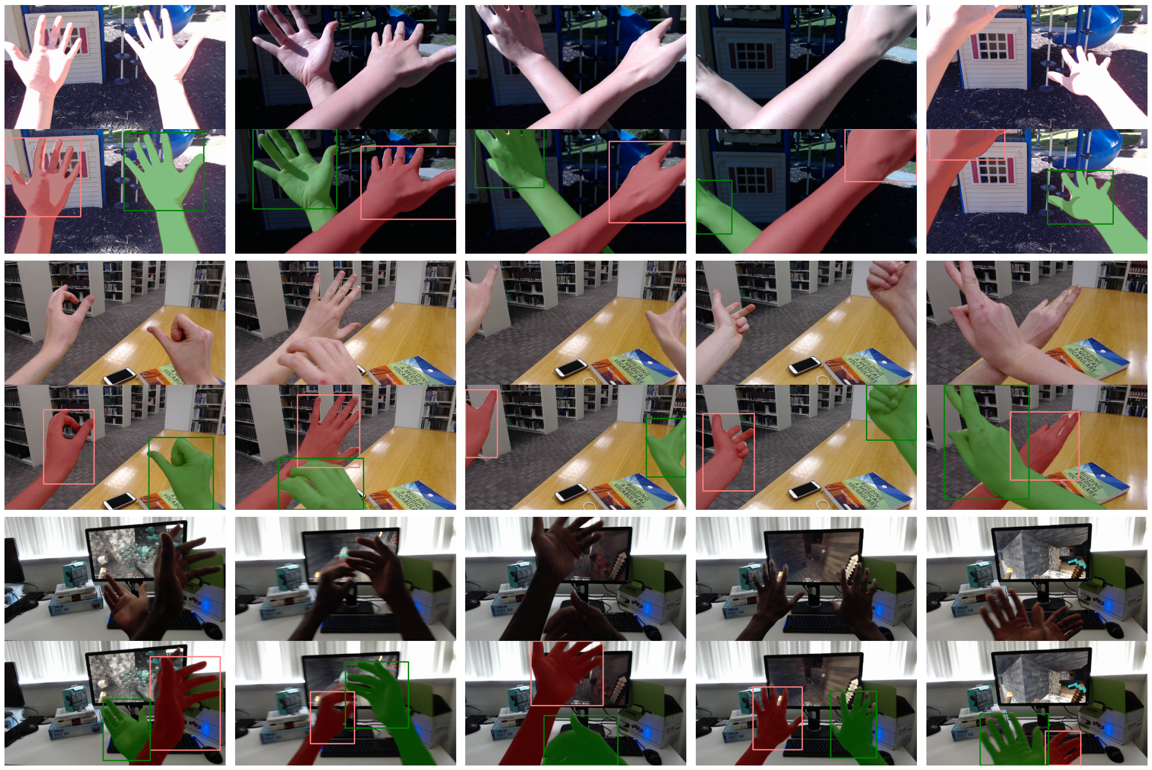


Figure 6: Qualitative results obtained using scene-adapted CSM-stage2. Odd rows show sample images in evaluation sequences with various skin tones and illumination. Even rows show the output visualization. **Best viewed in magnification.**

our revolutionary level of generalization, as [13] very recently tried to address the problem of domain adaptation in a specific unseen environment for binary-label hand segmentation, we enable models to achieve high accuracy on two-hand segmentation and detection in domain-invariant setting with the option to further improve using scene-specific adaptation. Qualitative results are provided in Fig. 6.

Conclusion

In this work, we introduce a color-based hand dataset and the corresponding training technique that helps deep convolutional networks to achieve domain generalization on the task of two-hand segmentation/detection. We propose a novel architecture CSM capable of producing high accuracy with compact model size and fast inference speed practical for real-world applications. Thorough evaluation and analysis for state-of-the-art models are reported on our new benchmark dataset. We hope our work can open up more directions for color-based hand tracking systems in the research community and industry.

REFERENCES

1. “Oculus Quest,” *https://www.oculus.com/quest/*, 2019.
2. “HTC Vive,” *https://developer.vive.com/resources/vive-sense/sdk/vive-hand-tracking-sdk/*, 2019.
3. “GoPro Camera Series,” *https://gopro.com/en/us/*, 2019.
4. “Narrative Clip,” *http://getnarrative.com/*, 2015.
5. J. Taylor, V. Tankovich, D. Tang, C. Keskin, D. Kim, P. Davidson, A. Kowdle, and S. Izadi, “Articulated Distance Fields for Ultra-Fast Tracking of Hands Interacting,” *In SIGGRAPH Asia*, 2017.
6. F. Mueller, M. Davis, F. Bernard, O. Sotnychenko, M. Verschoor, M. A. Otaduy, D. Casas, and C. Theobalt, “Real-time Pose and Shape Reconstruction of Two Interacting Hands With a Single Depth Camera,” *ACM Transactions on Graphics (TOG)*, 2019.
7. F. Lin, C. Wilhelm, and T. Martinez, “Two-hand Global 3D Pose Estimation Using Monocular RGB,” *arXiv preprint arXiv:2006.01320*, 2020.

8. S. Bambach, S. Lee, D. J. Crandall, and C. Yu, “Lending A Hand: Detecting Hands and Recognizing Activities in Complex Egocentric Interactions,” *In ICCV*, 2015.
9. A. U. Khan and A. Borji, “Analysis of Hand Segmentation in the Wild,” *In CVPR*, 2018.
10. Y. Li, Z. Ye, and J. M. Rehg, “Delving into Egocentric Actions,” *In CVPR*, 2015.
11. C. Li and K. M. Kitani, “Pixel-level Hand Detection in Ego-Centric Videos,” *In CVPR*, 2013.
12. Y. Li, M. Liu, and J. M. Rehg, “In the Eye of Beholder: Joint Learning of Gaze and Actions in First Person Video,” *In ECCV*, 2018.
13. M. Cai, F. Lu, and Y. Sato, “Generalizing Hand Segmentation in Egocentric Videos with Uncertainty-Guided Model Adaptation,” *In CVPR*, 2020.
14. K. Fukushima, “Neocognitron: A hierarchical neural network capable of visual pattern recognition,” *In Neural Networks*, vol. 1, pp. 119–130, 1988.
15. Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, “Object Recognition with Gradient-Based Learning,” *In Shape, contour and grouping in computer vision*, pp. 319–345, 1999.
16. K. Chellapilla, S. Puri, and P. Simard, “High performance convolutional neural networks for document processing,” *In Workshop on Frontiers in Handwriting Recognition*, 2006.
17. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” *In CVPR*, 2009.
18. A. Krizhevsky, I. Sutskever, and G. Hinton, “Imagenet classification with deep convolutional neural networks,” *In NIPS*, 2012.
19. K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *In CVPR*, 2016.
20. T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C. L. Zitnick, “Microsoft COCO: Common Objects in Context,” *In ECCV*, 2014.
21. J. Tompson, M. Stein, Y. Lecun, and K. Perlin, “Real-Time Continuous Pose Recovery of Human Hands Using Convolutional Networks,” *ACM Transactions on Graphics (TOG)*, 2014.
22. S. Sridhar, F. Mueller, A. Oulasvirta, and C. Theobalt, “Fast and Robust Hand Tracking Using Detection-Guided Optimization,” *In CVPR*, 2015.
23. T. Sharp, C. Keskin, D. Robertson, J. Taylor, J. Shotton, D. Kim, C. Rhemann, I. Leichter, A. Vinnikov, Y. Wei, D. Freedman, P. Kohli, E. Krupka, A. Fitzgibbon, and S. Izadi, “Accurate, Robust, and Flexible Real-time Hand Tracking,” *In CHI*, 2015.
24. A. K. Bojja, F. Mueller, S. R. Malireddi, M. Oberweger, V. Lepetit, C. Theobalt, K. M. Yi, and A. Tagliasacchi, “HandSeg: An Automatically Labeled Dataset for Hand Segmentation from Depth Images,” *arXiv preprint arXiv:1711.05944*, 2018.
25. G. M. Lim, P. Jatesiktat, C. W. K. Kuah, and W. T. Ang, “Hand and Object Segmentation from Depth Image using Fully Convolutional Network,” *In EMBC*, 2019.
26. X. Ren and C. Gu, “Figure-Ground Segmentation Improves Handled Object Recognition in Egocentric Video,” *In CVPR*, 2010.
27. A. Fathi, X. Ren, and J. M. Rehg, “Learning to Recognize Objects in Egocentric Activities,” *In CVPR*, 2011.
28. Y. Sheikh, O. Javed, and T. Kanade, “Background Subtraction for Freely Moving Cameras,” *In ICCV*, 2009.
29. A. A. Argyros and M. I. Lourakis, “Real-time Tracking of Multiple Skin-colored Objects with a Possibly Moving Camera,” *In ECCV*, 2004.
30. P. Kakumanu, S. Makrogiannis, and N. Bourbakis, “Real-time Tracking of Multiple Skin-colored Objects with a Possibly Moving Camera,” *Pattern Recognition*, vol. 40, pp. 1106–1122, 2007.
31. G. Serra, M. Camurri, L. Baraldi, M. Benedetti, and R. Cucchiara, “Hand Segmentation for Gesture Recognition in EGO-Vision,” *In IMMPD*, 2013.
32. C. Li and K. M. Kitani, “Model Recommendation with Virtual Probes for Egocentric Hand Detection,” *In ICCV*, 2013.
33. A. Betancourt, P. Morerio, E. Barakova, L. Marcenaro, M. Rauterberg, and C. Regazzoni, “Left / Right Hand Segmentation in Egocentric Videos,” *Computer Vision and Image Understanding*, vol. 154, pp. 73–81, 2016.
34. G. Lin, A. Milan, C. Shen, and I. Reid, “RefineNet: Multi-Path Refinement Networks for High-Resolution Semantic Segmentation,” *In CVPR*, 2017.
35. J. Wang, F. Mueller, F. Bernard, S. Sorli, O. Sotnychenko, N. Qian, M. A. Otaduy, D. Casas, and C. Theobalt, “RGB2Hands: Real-Time Tracking of 3D Hand Interactions from Monocular RGB Video,” *ACM Transactions on Graphics (TOG)*, vol. 39, 12 2020.
36. O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” *In MICCAI*, 2015.
37. M. Cai, K. Kitani, and Y. Sato, “An Ego-vision System for Hand Grasp Analysis,” *IEEE Transactions on Human-Machine Systems*, vol. 47, pp. 524–535, 8 2017.
38. I. M. Bullock, T. Feix, and A. M. Dollar, “The Yale human grasping dataset: Grasp, object, and task data in household and machine shop environments,” *The International Journal of Robotics Research*, vol. 34, 2015.

39. F. Perazzi, J. Pont-Tuset, B. McWilliams, L. Van Gool, M. Gross, and A. Sorkine-Hornung, "A Benchmark Dataset and Evaluation Methodology for Video Object Segmentation," *In CVPR*, 2016.
40. J. Pont-Tuset, F. Perazzi, S. Caelles, P. Arbeláez, A. Sorkine-Hornung, and L. Van Gool, "The 2017 DAVIS Challenge on Video Object Segmentation," *arXiv:1704.00675*, 2017.
41. M. Everingham, S. M. A. Eslami, L. V. Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes Challenge: A Retrospective," *International Journal of Computer Vision*, vol. 111, pp. 98–136, Jan. 2015.
42. F. Lin, C. Yao, and T. Martinez, "Flow Adaptive Video Object Segmentation," *Image and Vision Computing*, 2020.
43. S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh, "Convolutional Pose Machines," *In CVPR*, 2016.
44. W. Wang, K. Yu, J. Hugonot, P. Fua, and M. Salzmann, "Recurrent U-Net for Resource-Constrained Segmentation," *In ICCV*, 2019.
45. V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, pp. 2481–2495, 1 2017.
46. H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia, "ICNet for Real-Time Semantic Segmentation on High-Resolution Images," *In ECCV*, 2018.
47. L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation," *In ECCV*, 2018.

Alex Lin is a PhD student at Brigham Young University. His area of interest is computer vision and deep learning and their applications.

Game Theory: An Examination of Cooperation and Coalitions

A. S. Peterson
Brigham Young University

***Abstract*—Game theory has been an influential topic throughout history. In every stage of humanity, decisions rely upon decisions made by others. As civilization has grown to what we now know, game theory has evolved as well. It has taken a formal mathematical formulation. It has received generalized solutions to some of it's most important constructs of games. it has also developed a notion of coalitions and how they can cooperate to improve outcomes over what an individual can get alone. This concept of cooperative game theory and coalitions was birthed during the peak of game theory. That is, during the breakthrough's of game theory's founding fathers: John Von Neumann, John Nash, and many others.**

■ **TO UNDERSTAND** cooperative and coalitional game theory, we must first understand the foundations of where game theory arose. Game theory is an extensively studied area in many fields such as economics, biology, and politics. Game theory has been widely studied and its mathematical foundations and formulations are rather new, dating less than one hundred years ago.

Game theory's first general formulation was by John Von Neumann and Oskar Morgenstern in 1944, however the concept of game theory came long before Neumann and Morgenstern defined it. Throughout history, decisions made are directly influenced by thinking about the potential decisions of others. This concept is common throughout human history, where individuals were passively applying all of the concepts of game theory. Historically popular names such as the Spanish conqueror Cortez, William Shakespeare, and Thomas Hobbes are among a few to display game theory before its formal introduction to science.

When the Spanish conqueror Cortez first landed in Mexico, he was met by the indigenous Aztecs. The Aztecs were far more numerous than Cortez' men, and thus Cortez' men were outmatched. In order to keep his troops from making the rational decision of retreating, Cortez burned all of his ships, thus changing the rational decision of his troops to fight for their lives. Additionally, this had an intimidating effect on the Aztecs, as their rational thinking led them to believe that anyone who would burn their own ships must be so sure of victory that it would be unwise to fight them. This caused the Aztecs to retreat into the hills, giving Cortez the best possible outcome. [1]

In William Shakespeare's *Henry V*, Henry decides to kill his French prisoners, in full vision of the enemy and Henry's men. This action for Henry's own men is a metaphorical ship burning, signifying that death awaits them if they do not prevail against the enemy. For the Enemy this affects their view of the potential payoffs, making the payoff for losing much worse than what they may have thought prior. This action changed the incentives for both sides in a way that benefited Henry and the English's prospects of victory.[1]

Philosopher and author of The Leviathan, Hobbes, thought that the ideal situation is for all to be able to act freely. Cooperation is an obvious choice for some, as they can achieve more together than without one another, but

Figure 1. William Shakespeare's *Henry V* is one of the first examples of game theory in literature.

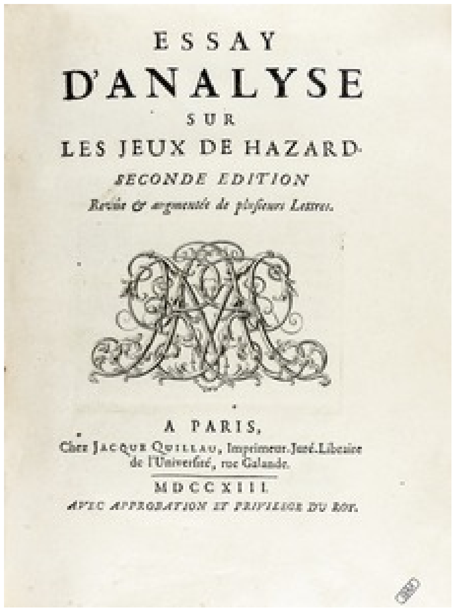
problems arise. This exists with immoral and amoral agents. Rational thinkers will question if the other party is going to return the promises of the relationship. This will lead to preemptive thinking, hurting the other before they themselves could be harmed. Eventually this cycle will lead to what Hobbes called a "war of all against all". The only way to combat this is to lead the people under a tyranny or anarchy. These political outcomes offer punishments for not keeping these cooperative promises, and create social dilemmas for participants. Thus we see how cooperation can be enforced by what we will call "contracts". This example from the Leviathan shows us how cooperative game theory can differ from non-cooperative game theory. Before exploring the roots of specifically cooperative game theory and coalitional game theory, we need to understand the foundations of game theory in general. [1]

ROOTS OF GAME THEORY

Waldegrave's Problem

The formal formulation of game theory began prior to Neumann and Morgenstern's seminal paper. Pierre Rédmond de Montmort was the author of the famous Waldegrave problem. The problem was established first on record in April 1711, from Montmort to Nicholas Bernoulli. The Waldegrave problem is the first appearance of a mixed strategy solution in game theory.[2]

The description of the game is as follows: Suppose there are n+1 players with each putting



ESSAY
D'ANALYSE
SUR
LES JEUX DE HAZARD.
SECONDE EDITION
Revûe & augmentée de plusieurs Lettres.

A PARIS,
Chez JACQUE QUILLAU, Imprimeur-Juré-Libraire de l'Université, rue Galande.
MDCCXIII.
AVEC APPROBATION ET PRIVILEGE DU ROY.

wrapfig

Figure 2. Pierre Rédmond de Montmort's book in which the waldegrave problem is published

one unit into the pot. The first two players play each other and the winner plays the third player. The loser of each game puts one unit into the pot. Play continues until one player has beaten all of the others in succession. The problem according to Montmort, is to find the expectation of each player and the probability that the pot will be won within a specified number of games. Montmort's Waldegrave problem represents the first formal thinking on the concept of game theory while associated with one of the pillars of statistics, Nicholas Bernoulli. Montmort also published this problem in his famous book of probability and games of chance, titled, *Essay d'analyse sur les jeux de hazard*.[2]

Edgeworth

Francis Ysidro Edgeworth published an 1881 paper titled, *Mathematical Physics: An Essay on the Application of Mathematics to the Moral Sciences*. In this publication Edgeworth considered the problem of determining the outcome of trading between individuals. In solution to this problem Edgeworth discovered what he coins the contract curve. In a game containing two commodities and two types of consumers, he demonstrates how his contract curve shrinks to the set. Edgeworth's contract curve developed into the idea of the core. The core can be defined as a set of payoff allocations $x \in R^N$ satisfying

1) Efficiency: $\sum_{i \in N} x_i = v(N)$
2) Coalitional rationality: $\sum_{i \in C} y_i \geq v(C)$ for all coalitions $C \subseteq N$

The core can be simply defined to be the set of imputations that are not dominated, or the set of allocations that cannot be improved upon by changing coalitions. This turns out to be one of a few famous solutions to a cooperative game. The core is among the first formulated mathematical structures for cooperative game theory before its formal introduction by Neumann and Morgenstern.

Émile Borel

In 1938, Émile Borel published a book titled, *Applications aux Jeux de Hasard* . In this book Borel proves a minimax theorem for two-person zero-sum matrix games when the payoff matrix is symmetric and also to a specific game called Blotto. Although this was a great accomplishment it still lacked the general ability to be applied to other problems. [3]

John Von Neumann

John Von Neumann is most known for his and Morgenstern's 1944 *"Theory of Game and Economic Behavior"*, however, his inclusion in game theory began much earlier. In 1928, Neumann proved the minimax theorem which states that every two-person, zero-sum game with finitely many strategies for each player is determined. [4]

The foundations of Montmort lead to the seminal paper by John Von Neumann and Oskar Morgenstern. In their 1944 paper titled, *"Theory of Game and Economic Behavior"*, non-cooperative game theory and cooperative game theory receive formal definitions. Neumann and Morgenstern define several important areas of cooperative game theory.

Figure 3. Jon Von Neumann details the first mathematical formulation of game theory [4]

Utility

An important area of game theory is utility. Neumann and Morgenstern define utility as "the fundamental concept of individual preferences" (p.15). In 1938 Paul Samuelson created a mathematical foundation for defining a utility function and in 1944 Neumann and Morgenstern provide an axiomatic theory of utility using Bernoulli's old theory of utility as an independent discipline. Neumann and Morgenstern's axioms are as follows:

Neumann and Morgenstern define a lottery L as probabilities that mutually exclusive events will happen, all probabilities summing to 1.

1) Axiom 1 (completeness)
   For lotteries L, M exactly one of the following holds:
   $L < M, M < L$, or $L \sim M$
2) Axiom 2 (transitivity)
   If $L < M$ and $M < N$, then $L < N$
3) Axiom 3 (continuity)

If $L \leq M \leq N$, then there exists a probability $P \in [0,1]$such that
$pL + (1-p)N \sim M$

4) Axiom 4 (independence)
For any N and $p \in (0,1]$, $L \leq M\ iff$
$pL + (1-p)N \leq pM + (1-p)N$

Neumann and Morgenstern propose that any set L that satisfies these axioms is a utility function. One might then ask, what happens if coalitions can split the payoffs however they desire? This would then open the door for bargaining to be in a group as it could be better than the Nash Equilibrium, or the stable state of no cooperation. In order to allow this kind of bargaining and cooperation, we must have transferable utility. [4]

Transferable Utility

In 1991, Roger B Myerson stated that transferable utility is the idea that given a coalition C, the payoffs are given by coalition rather than individual, where it is then the responsibility of the coalition to disburse the payoff to the individuals. Such transfers are possible if the players have a common currency that is valued equally by all. Transferable utility is an assumption of cooperative game theory as it allows for there to be negotiation and coalition forming based on how the payoff will be split after the whole coalitions payoff (which is larger than each player would have had individually) is received. With the idea of transferable utility in mind, we can now explore how a bargaining game was created through this concept. [3]

Bargaining

Bargaining is an idea first seen in the seminal work Neumann and Morgenstern. Here they proposed that in a two-person game, the two players could be allowed to communicate and negotiate. This takes the turn towards cooperative game theory, however bargaining games are only cooperative to the extent that the agreements are able to be enforced. A bargaining game is set up to be a noncooperative game where two players negotiate on the division of a surplus known as the alternating offers bargaining game. Players take turns being the proposer of the game. Division of surplus in unique subgame perfect equilibrium depends upon how strongly players prefer current over future payoffs. As players become perfectly patient in this game, the equilibrium converges to the Nash bargaining solution. This two-person bargaining game consists of a feasibility set F, a closed subset of R^2 which is assumed to be a convex set of agreements. F is assumed convex because for any two feasible outcomes a convex combination, or weighted average of them is also feasible. A two-person bargaining game also consists of a disagreement point $d = (d_1, d_2)$, where d_1 and d_2 are the payoffs to player 1 and player 2 respectively. This is the payoff that each player is guaranteed if no agreement can be made. There are some different solutions that have been proposed to bargaining problems, among which, the most well known is the Nash bargaining solution. [3]

Figure 4. John Nash Jr. developed a solution to the bargaining problem, leading to cooperative game theory. [5]

Nash Bargaining Solution

In 1950, John Nash proposed an axiomatical solution to the bargaining problem in his work titled, *The Bargaining Problem*. In classical Nash style, this work is only 8 pages long yet provides a compelling solution to the bargaining problem

[5]. Nash states that Neumann and Morgenstern proposed a theory that n-person games include a special case, the two-person bargaining problem. However as Nash states, "the theory there developed makes no attempt to find a value for a given n-person game, that is, to determine what it is worth to each player to have the opportunity to engage in the game ... It is our viewpoint that these n-person games should have values" [5]. Nash saw that, although Neumann and Morgenstern covered a large portion of game theory, there was a need for a solution to this two-person bargaining problem. Before we get to Nash's axioms for the bargaining solution, Nash proposed that we will only be able to provide a utility function for our two players if the following criteria hold:

1) An individual offered two possible anticipations can decide which is preferable or that they are equally desirable.
2) The ordering thus produced is transitivee; if A is better than B and B is better than C then A is better than C.
3) Any probability combination of equally desirable states is just as desirable as either.
4) If A, B, and C are as in assumption (2), then there is a probability combination of A and C which is just as desirable as C. This amounts to an assumption of continuity.
5) . If $0 \leq p \leq 1$ and A and B are equally desirable, then $pA + (1 - p)C$ and $pB + (1 - p)C$ are equally desirable. Also, if A and B are equally desirable, A may be substituted for B in any desirability ordering relationship satisfied by B.

Now that Nash has defined the fundamentals of what assumptions must be met in order to have utility functions for the players of the game, he proposes that a solution should satisfy the following axioms:

1) Invariant to equivalent utility representations (affine transformations).
2) Pareto optimality.
3) Independence of irrelevant alternatives.
4) Symmetry.

And thus, Nash proved that the solutions satisfying these axioms are the points (x, y) in F which maximize:

$$(u(x) - u(d))(v(y) - v(d)) \tag{1}$$

where u and v are the respective utility functions of player 1 and player 2, and d is the disagreement point. $u(d)$ and $v(d)$ are called the status quo utilities, or the utilities given if one player decides not to bargain. The product of the two excess utilities is called the *Nash product*. It is intuitive to see that the Nash bargaining solution gives each player their status quo as well as a share of the payoff that comes from cooperation. However this is still a noncooperative game, which leads us to another key aspect of cooperative game theory, that is coalitions. [5]

Coalitions

In our Nash bargaining solution above, one can see the principals of cooperation building in game theory. However the game still consists of two players pitted against each other. What happens when the number of players start to increase in a game that rewards cooperation? There are games out there that, either naturally or artificially, incentivize the forming of groups among players. These groups take on a formal name of coalitions. [3]

SEPARATION FROM NON-COOPERATIVE GAME THEORY

The difference between non-cooperative game theory and cooperative game theory can get very unclear at times, when in non-cooperative game theory there can be bargaining which is a form of cooperation. So where exactly is the line? In his book *Game Theory: Analysis of Conflict*, Roger B. Myerson says, "The key assumption that distinguishes cooperative game theory from noncooperative game theory is this assumption that players can negotiate effectively." [3] (p.419-p.420) In a cooperative game, competition among coalitions is incentivized by the existence of external enforcement, or contract law. In noncooperative games there are either no alliances allowed or they are strictly self-enforcing (i.e. through rational threats). This type of coalitional game is not considered a cooperative game but as one can easily see, it most certainly has characteristics of a cooperative game. [3]

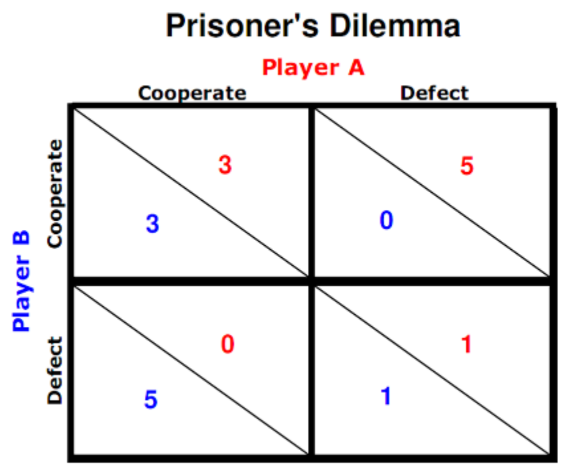


Figure 5. Prisoners dilemma. Cooperate corresponds to a participant choosing to talk. Defect corresponds to a participant choosing to remain silent.

Cooperative game theory makes some key assumptions that define its nature. These assumptions are the following:

1) Players can negotiate effectively
2) Transferable utility

Cooperation in The Prisoners Dilemma

An illustration of why cooperation can lead to better payoffs than Nash Equilibrium can be shown easily in the prisoners dilemma (Figure 3). The prisoners dilemma says that two people (players) are being held in separate rooms. Each player has been given the following options: Defect (tell the interrogator nothing) or Cooperate (tell the interrogator what you know). The payoff matrix in Figure 3 shows that our payoff relies directly on what the other player chooses, however there is no way to talk to the other player to come up with a solution. The Nash Equilibrium of this problem is both player A and player B choosing to choose cooperate which gets both players three years of jail time. This is because if either player were to switch their choice with the other player staying the same, their payoff would decrease. Therefore this is a stable, Nash Equilibrium. However as you may have noticed, what would happen if the two players could communicate? They may come to the agreement that if they both choose to defect then both of them will only have to serve one year rather than the alternative of three or five.[3]

This example shows how cooperating or bargaining can lead to better payoffs for all players, which leads us to the concept of cooperative game theory. Cooperative game theory, rather than having two players, has players who are allowed or incentivized to create groups/coalitions. Contracting as previously stated is key to the differentiation of cooperative game theory from noncooperative. Why the need for such contracting at all? An experiment conducted by Reinhard Selten can help us understand more deeply.[3]

Selten details his experiment in his 1986 manuscript, *End Behavior in Sequences of Finite Prisoner's Dilemma Supergames*. In this work, Selten conducts a game where participants played a repeated Prisoner's Dilemma with real money. Player's did not know how many repetitions there would be, but they knew it could not last longer than a specified amount of time. The participants were allowed to communicate which throughout the majority of the game resulted in the players acting as we decided would be optimal in the previous paragraph, but without enforcement of their agreement, as the clock got closer to the end of the game players were incentivized to switch to get a better payout while hurting the other player. This shows how to informal "contract" of negotiating to an optimal for both players may only last until the rational threat of the other player not trusting them or turning on them will only last until that threat is no longer valid. In our case the threat was no longer valid as the experiment was coming to an end so the player thought that it did not matter if the other player trusts them anymore. And thus we see that there is a good and valid place for cooperative games as noncooperative games can have some undesirable outcomes in certain situations.[3]

ORIGINS OF THE PILLARS OF COOPERATIVE GAME THEORY

Neumann and Morgenstern propose a game that introduces cooperative and coalitional game theory. The game is as follows; There are three players, where the set of strategy options for each player i is

$$C_i = (x_1, x_2, x_3) \in R^3 | x_1 + x_2 + x_3 \leq 300 \tag{2}$$

and $x_j \geq 0$

Thus each player can propose a payoff allocation for the three players such that no player's payoff is negative and the sum of their payoff does not exceed 300. Our utility function is

$$u_i(c_1, c_2, c_3) = x_i \tag{3}$$

if $c_j = c_k = (x_1, x_2, x_3)$ for some $j \neq k$

$$u_i(c_1, c_2, c_3) = 0 \tag{4}$$

if $c_1 \neq c_2 \neq c_3 \neq c_1$

Assuming that the players can negotiate effectively, this simple game becomes quite complicated. They propose that players 1 and 2 decide to negotiate to form a coalition and split the profit 50/50. But then player 3 decides that getting an unfair split is much better than nothing, so they offer player 1 to split 60/40 thus providing incentive for player 1 to switch to form a coalition with player 3. However this cycle continues, and how does one come to a solution? It never does, that is, without a change in the game to restrict number of times one can switch coalitions, a time limit, etc.. With a restriction such as number of times a player can switch coalitions, Neumann and Morgenstern propose that the characteristic function has the general form:

$$v(S) = min_{\sigma_{N/S} \in \Delta(C_{N/S})} max_{\sigma_S \in \Delta(C_S)}$$

$$\sum_{i \in S} u_i(\sigma_s, \sigma_{N/S}) \tag{5}$$

where N/S denotes the set of all players in N who are not in the coalition S. We let $u_i(\sigma_s, \sigma_N/S)$ signify player i's expected payoff, before transfers of money, when the correlated strategies σ_S and $sigma_{N/S}$ are independently implemented. That is,

$$u_i(\sigma_s, \sigma_{N/S}) =$$

$$\sum_{c_s \in C_s} \sum_{c_{N/S} \in C_{N/S}} \sigma_S(c_S) \sigma_{N/S}(c_{N/S}) u_i(c_S, c_{N/S}) \tag{6}$$

This asserts that $v(S)$ is the maximum sum of utility payoffs that members of coalition S can promise themselves. If this is satisfied, then we say that v is the *minimax representation*.

Figure 6. In 1953, Lloyd Shapley formulates a way to allocate payoffs for a coalition to its members based on their contribution to the coalition. [6]

Shapley Value

In cooperative game theory, where a game involves coalitions of more than one player, there are some basic parts. The first part is a utility function for how much payoff a player can receive on their own as well as utility for each coalition. Once a player knows how much each coalition will make as a whole they can make a decision of what coalitions to form. However there may be something more important to consider when choosing which coalitions to form, that being the individual payoff after splitting within the coalition. One would expect individual payoff to directly correlate to the individuals contribution to the coalition, but how does one accomplish this allocation of payoffs? One possible answer to this question is offered by Lloyd Shapley in 1951. Shapley's suggestion holds for the following definition of a coalitional game. There is a set N of n players and a function v that maps subsets of players to the reals: $v : 2^N \rightarrow R$ with $v(\emptyset) = 0$ where $\emptyset$ denotes the empty set and v is a characteristic function. $v(S)$ is called the worth of the coalition S defined as the total payoff received by the coalition. According to the Shapley value, the amount given to player i is:

$$\varphi_i(v) = \sum_{S \subseteq N-i} \frac{|S|!(n-|S|-1)!}{n!} * (v(S \cup i) - v(S)) \tag{7}$$

where n is the total number of players and the and the sum extends over all subsets S of N not containing player i. In his book, Roger Myerson describes the interpretation of the Shapley value as follows. "Suppose that we plan to assemble the grand coalition in a room, but the door to the room is only large enough for one player to enter at a time, so the players randomly line up in a queue at the door. There are $|N|!$ different ways that the players might be ordered in this queue. For any set S that does not contain player i, there are $|S|!(|N|-|S|-1)!$ different ways to order the players so that S is the set of players who are ahead of player i in the queue. Thus, if the various orderings are equally likely,

$$|S|!(|N|-|S|-1)!/|N|! \tag{8}$$

is the probability that, when player i enters the room, he will find the coalition S there ahead of him. If i finds S ahead of him when he enters, then his marginal contribution to the worth of the coalition in the room when he enters is $v(S \cup i) - v(S)$. Thus, under this story of randomly ordered entry, the Shapley value of any player is his expected marginal contribution when he enters.". In a 1974 book by Shapley and Robert Aumann, they extend the Shapley value with respect to infinite games as well. [6]

CONCLUSION

Cooperative game theory was created during a time with many brilliant minds working together to figure out how to mathematically formulate cooperation and its benefits. The birth of game theory and cooperative game theory happened concurrently, thus the distinction of when cooperative game theory was created can be grey. Cooperative game theory truly is the result of cooperation among brilliant mathematicians and economists.

ACKNOWLEDGEMENT

I would like to thank Sean Warnick for his help in the development of my interest in this topic.

REFERENCES

1. D. Ross, "Game theory," Mar 2019.
2. D. Bellhouse, "The problem of waldegrave," *Journal Électronique d'Histoire des Probabilités et de la Statistique [electronic only]*, vol. 3, 01 2007.
3. R. B. MYERSON, *Game Theory: Analysis of Conflict*. Harvard University Press, 1991.
4. J. Von Neumann and O. Morgenstern, "Theory of games and economic behavior," *Princeton University Press*, 1944.
5. J. F. Nash, "Equilibrium points in n-person games," *Proceedings of the National Academy of Sciences*, vol. 36, no. 1, pp. 48–49, 1950.
6. L. S. Shapley, *A Value for n-Person Games*. Santa Monica, CA: RAND Corporation, 1952.

Alex Peterson is a Masters student who has been with IDeA Labs since fall of 2020. He comes from an undergraduate degree in Statistics from the University of Missouri and is looking to combine his knowledge of statistics with optimization, learning, and control processes. His research experience is in ecological population modeling while his research interests span the crossroads of economics, engineering systems, and data science. His hobbies include creating mobile applications, fishing, playing games with his nieces and nephews, and finding new restaurants with his wife.

Multi-Agent Learning in Markov Games

Tim Whiting
Brigham Young University

***Abstract*—Strategic planning plays a vital role in our everyday life. We use it to determine the best time to wake up, which tasks to prioritize today, what to eat, and many other daily decisions. In a more explicit way, strategy defines human interactions in many varieties of games. Game theory is the mathematical foundation for why we make the choices we do. Game theory has been formally studied by influential minds since the early 1900s, but only in the last few decades has it been used as a basis for automated agents to reason about the complexities of the world, mainly because it took that long to develop the requisite computational power. For a long time, game theory was relegated to games, economics, and political strategy, but now it is becoming essential in our daily interactions with smart devices. Presented are the influential people and ideas that have led to the innovation of computer agents that interact strategically with other rational agents as well as humans. In this journey through time, important concepts in game theory and important properties of game theoretic agents will be highlighted, along with the influential people who worked to make these concepts and properties a reality.**

■ "**WHY SHOULD I CARE** about game theoretic agents?" you might ask. Well, for one thing, they care a lot about you. They analyze your moves, watch your choices, and make decisions that impact your life in a major way. You are being manipulated or observed by strategic agents almost every time you go online. For example, recommendation systems analyze the statistics of your actions, and neural networks identify patterns in your behavior. However, what if they could go one step farther, and play you like they play a game of chess? Would your actions truly be your own anymore? Game theory, at its heart, tries to understand why the choices we make matter, how others' choices influence us, and what are the idealistic, optimal, safe, or sane choices to make. All choices have consequences, and even the smallest of those consequences are important to you because they affect your life directly.

Agents

Agents are an encoding of a particular strategy that reasons about and manipulates actions and consequences. This encoding is called a policy. A human or animal agent processes potential actions and their consequences using a brain. As humans, we rely on abstractions, assumptions, and generalizations to weigh different options, the cost or benefit of each action, and whether they align with our priorities or meet our goals. In a very similar way, agents also rely on abstractions, assumptions, and generalizations; however, their ability to generalize comes with a price. Generalizations don't always work well for extended periods of time, or in different situations. Nevertheless, they do often work, and there are many theorems showing why and where certain assumptions and generalizations hold true. This paper explores fundamental concepts in game theory and identifies important properties that are commonly used in the creation of game-theoretic

Table 1. A Sample Matrix Game

Player		Column	
	Actions	X	Y
Row	A	10, 2	20, 10
	B	0, 20	15, 10

A sample matrix game. Row has actions A and B available and Column has actions X and Y. Row player's utility is on the left, Column's utility is on the right. This sample illustrates the fact that games don't have to be zero-sum.

Table 2. Prisoner's Dilemma

Player		Column	
	Actions	Deviate	Cooperate
Row	Deviate	-2, -2	-3, 0
	Cooperate	0, -3	-1, -1

The prisoner's dilemma. This matrix game is a simultaneous action game, so both players must choose actions prior to knowing what the other player chose. The best joint utility is for both players to cooperate. Self-interested agents can do better by deviating when the other player chooses cooperate. However, if both players deviate, they both do worse.

Table 3. Matching Pennies

Player		Column	
	Actions	Heads	Tails
Row	Heads	1, 1	-1, -1
	Tails	-1, -1	1, 1

The game Matching Pennies. The players earn money by coordinating to play heads or tails, thus matching their strategies.

Table 4. Chicken

Player		Column	
	Actions	Swerve	Straight
Row	Swerve	0, 0	-1, 1
	Straight	1, -1	-100, -100

The game Chicken. There is a lot of motivation to avoid crashing, and thus to cooperate by coordinating opposing strategies. Either both players have to choose to swerve, or only one or the other player does well.

agents.

Game Theory

Game Theory is a logical system of analysis of interactions between agents who make rational choices based on actions of themselves, actions of others, and possible outcomes. Important in this definition is the fact that the choices must be rational, and the agents must assign some value to possible outcomes. If the agents violate these assumptions, the theorems and definitions are assumed to be invalid, even though they might hold true in certain cases.

Games

In order to discuss agents, we must first discuss games and strategies. We study games because they model the interaction between multiple parties who are not necessarily on the same page. Games also involve strategy and reasoning, and provide an interesting subject to analyze how math can be used to help computers 'reason'.

There are many types of games, and each game can be classified in many different ways. However, one thing they have in common is the notion of actions and consequences. At distinct points in a game, there are a set of actions available to both players, and consequences for those actions. The consequences might have short-term or long-term effects. The consequences of games are typically represented by a set of utilities denoted u_i for the agent's utility, and u_{-i} models opposing players utility in two player games (u_k for 3+ player games). This notion of actions and consequences, with utility tied to the consequences of one's actions, is familiar because we deal with such strategic choices every day. This is why games provide a good model to evaluate and test agents. Games can be made to be arbitrarily simple or complex to model different kinds of interactions or situations that an agent in the real world might have to deal with.

Two other classifications of games are important for our purposes: normal-form vs extensive-form games, and non-cooperative vs cooperative games. Normal-form games are simplified games that specify actions and consequences in a form of a table. They have been the most extensively studied. Extensive-form games, on the other hand, model more complex aspects of games such as imperfect information and sequencing of moves. Cooperative games allow alliances between groups of people, whereas non-cooperative games do not permit coalitions to form. In this article, we will explore the foundations of game theoretic agents in normal-form, non-cooperative games.

Normal-Form Games

Normal-form games are games with a single state S and complete specification of payoffs to each player for each strategy they can choose.

Table 5. Rock, Paper, Scissors

Player		Column		
	Actions	Rock	Paper	Scissors
Row	Rock	0, 0	-1, 1	1, -1
	Paper	1, -1	0, 0	-1, 1
	Scissors	-1, 1	1, -1	0, 0

The classic game of Rock, Paper, Scissors. Both players can either choose rock, paper, or scissors. If they choose the same action, the outcome is a draw. For each action there is one action it wins against and one action it loses against.

There are a few common normal-form games that are important to understanding and discussing non-cooperative game theory. I will represent them in the common format of a matrix game, where the actions are enumerated for each player: the 'Row' player abbreviated R and the 'Column' player abbreviated C. The rewards for each joint action are specified in the matrix entry for that joint action, with the row player's reward on the left, and the column player's reward on the right. An example matrix game is shown in Table 1.

First and foremost is the prisoner's dilemma, represented in Table 2. In this game, the predicament is that in order for both players to do well, they must trust the other player not to defect. However, if they choose that action, the rational choice for the other self-interested agent would be to defect and do better off, which in turn hurts their opponent.

Other important games for understanding the complexities of strategies and learning are the games of Matching Pennies, Chicken, and Rock, Paper, Scissors. These are shown in Tables 3, 4, and 5 respectively. Each of these games has a unique strategic issue at play. Some games, like Matching Pennies or the prisoner's dilemma, are exercises in cooperation. Other games, such as Chicken, are exercises in coordination. Still others, such as Rock, Paper, Scissors, seem to be based off luck or random chance. With learning agents, these games are typically played for a predetermined number of rounds, to allow time for learning and reacting to the other player's strategy and to see what the convergence, long-term strategies, and behavior of the agents are.

Non-Cooperative Games

In order to understand the foundations of Markov Game Theory and learning agents, we first must discuss the strategy space of non-cooperative games. The most basic assumption of a non-cooperative game is that the agents are in competition with each other, either because of the lack of actions that could lead to cooperation, such as in the game of Rock, Paper, Scissors, or because of a lack of ability to communicate or coordinate. Coordination is different from cooperation in the fact that you can coordinate movements to achieve outcomes that are not necessarily optimal in terms of cooperation. Two kinds of games often discussed are zero-sum games and general-sum games. A zero-sum game is one where one player's score is just the negation of the other player's score, so their scores sum to zero. A general-sum game can have some outcomes that sum to zero, but generally they are more flexible, so that multiple players can benefit from a single round. Rock, Paper, Scissors, chess, and other common games with a win-lose outcome are zero-sum games. General-sum games are more likely to occur in the real world scenarios, since the rewards are more nuanced than just a win-loss situation.

Strategies

There are two types of strategies that are typically discussed in game theory: mixed strategies and pure strategies. Strategies that are modeled by a probability distribution over the action space are called mixed strategies or strategy profiles. A strategy where one action is chosen with probability of 1 and all other actions have probability of 0 is called a pure strategy, but it is actually just a special case of a mixed strategy. Many people have been influential in our understanding the characteristics of strategies in non-cooperative games, most notably John von Neumann, Vilfredo Pareto, and John Nash.

John von Neumann

One of the common strategies used in games is called the MaxMin (or maximin) strategy, which is the assumption that the other player is in opposition to you. Proven in 1928 by the inventor and mathematician John von Neumann (shown in Figure 1) in his paper *Zur Theorie der Gesellschaftsspiele*[1], the idea of this strategy is to put a bound on the worst-case outcomes,

Figure 1. Mathematician and Inventor John von Neumann. John was influential in the development of the computer and the field of game theory. His proof of the maximin theorem in 1928 helped serve as a foundation for understanding strategy and subsequent Folk Theorems.

maximize the possible points you could achieve, and simultaneously minimize the points of the opposing player. The strategy is computed by determining the action you can take that maximizes your outcome, by assuming that your opponent will take the action that will minimize your outcome. This reasoning is easily modeled by a recursive algorithm on a game tree in turn-taking games. However, in simultaneous-action games, the reasoning is a little harder due to the stochastic nature of the outcomes. For example, if the Row player takes action A, the outcome will vary depending on the choice of the Column player, especially if there is no clear choice she would make knowing your outcomes and preferences. She could choose a mixed strategy, which would mean that the outcome is best modeled by a probability distribution over the actions that both you and the opponent choose. The solution in a simultaneous-action game can be solved by a linear program[2].

Unfortunately, the MaxMin strategy has a problem: it is inherently pessimistic. For example, in the game of Chicken, it will always choose the *swerve* action since it assumes the other player will always try to minimize his payoff. There are other strategies that result in solutions that can be at least as good as a MaxMin strategy. Let us talk about the people behind these ideas and the properties of the solution profiles they were interested in.

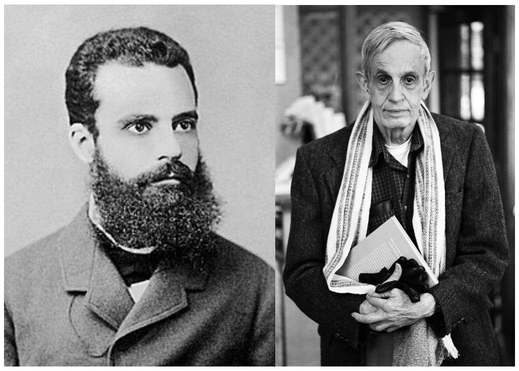


Figure 2. Economists and mathematicians Vilfredo Pareto (left) and John Nash (right). Each had a major impact on economic and game theory, influencing how we understand and talk about equilibrium strategies in non-cooperative games.

Vilfredo Pareto

In 1906, Vilfredo Pareto (shown in Figure 2) published his Manual of Political Economy[3], [4]. In it, he reported his findings about how there are certain places of equilibrium in society where there is a local optimality. In other words, there is no way to move from that position without increasing or decreasing the welfare of all people. Thus, if someone benefits, another person will have lost something. This is not the same thing as a zero-sum game, since both people could have a positive outcome still, it is just about the change in welfare. This principle is easily seen in many games. There are often multiple places of equilibrium in games where, if the players are in a certain situation, they will not both choose different situation. For example, in the game of Chicken, if Row chooses the action *straight* and Column chooses *swerve*, they are in a situation where the Row player would not choose to switch to any other outcome, even though Column would prefer a different outcome. The same reasoning follows for if the player's actions were reversed. These solutions are called Pareto optimal solutions of the game, in honor of Vilfredo Pareto.

John Nash

John Nash (shown in Figure 2), who won a Nobel prize in economics for his work on Game Theory, thought about the theory and mathematics

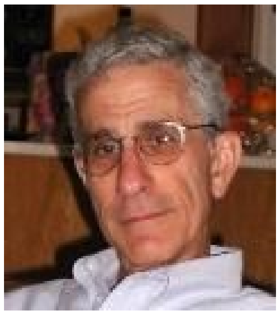


Figure 3. In 1971, economist James Friedman published his paper *A non-cooperative equilibrium for supergames*. These equilibrium strategies were commonly known but never formally published until he showed their existence in his proof. This strengthened the result of the original Folk Theorem which was widely known but never published.

behind non-cooperative games at Princeton University during his PhD. A mathematical genius, he solved the problem of finding equilibrium points in N-Person games [5] as part of his dissertation on non-cooperative games in 1950. These points are strategy profiles where neither player would prefer to switch to a different strategy profile unilaterally. It's important to note the differences between Nash equilibria and Pareto optimal solutions. Nash equilibria only consider one person being able to change their decision, whereas Pareto optimal solutions consider all other potential payoff combinations. For determining if a point is a Nash equilibria, only one person has to see something more desirable, whereas with Pareto optimal, both players have to agree that some point is better for both of them. Interestingly, in the prisoner's dilemma game, the Pareto optimal solutions are the three solutions where one or both people cooperate, whereas the Nash equilibrium is the solution where both deviate.

Folk Theorem

In 1971, James Friedman (shown in Figure 3) built off the principles discussed in the previous section and proved that any payoff above the MinMax value[1] for both players could be sustained as a Nash equilibrium of an infinitely repeated game[6]. The result relies on the fact that a player can hold a threat of punishment over another player if they do not stick to the equilibrium strategy. The player can thus drive the payoffs to at least the minimax value, and can go even further by not being rational, reducing both players' scores. Although most games we play aren't repeated infinitely, many do have rounds, and not letting the players know the number of rounds has a similar effect.

[1]the payoffs of the maximin strategy for each respective player

Learning Agents

Knowing the aforementioned strategies does help in designing an agent. However, there are many reasons why that knowledge is not enough. For example, even though we can guarantee a minimax outcome in a game, a good agent should be able to do better than obtain its best worse-case outcome all of the time. Ideally, it should do better than that most of the time. Also, not every opponent will be completely rational, nor will it be able to compute the full game. For humans, this is especially pertinent, since it is hard for us to think several moves in advance. An agent should be able to learn based off the common actions of its opponent, and then apply those strategies to improve its performance in the future. Also, for complex games, it is intractable to compute some of these solutions, and thus the agent needs to be able to reason about the game in an abstract way.

Markov Decision Processes

To understand this new representation of games for learning agents, we must first understand what a Markov Decision Process (MDP) is. In 1957, Richard Bellman (shown in Figure 4), then at The Rand Cooperation, described non-linear recurrence relations and a way to solve them using what is now known as 'value iteration'[7]. These non-linear recurrence relations described previously by Russian mathematician Andrey Markov say that each state can be described as a transition from a previous state. They are recurrent because the current state depends on all past states through a recurrence relation through the transition function, and doesn't depend on past states directly. Value Iteration shows how to 'solve' these recurrence relations to find optimal properties. A common formulation of the problem is as follows: Take Q to be the property that can be described through this recurrence relation, S to be the set of all states, A to be the set of all actions, T to be the transition function that specifies the probability of being in a new state s' given a particular state action pair (s, a), and R to be the instantaneous value of Q for that state, dependant on a particular state action pair (s, a). In addition, take γ as a discount factor based off the weighting of instantaneous versus long-term value of Q. Then the value of Q for a particular state action pair (s, a) can be specified as the following recurrence relation that will converge depending on the value of γ:

$$Q_{k+1}(s,a) = R(s,a) + \gamma * \sum_{s'} T(s',s,a) * V_k(s'),\ \ for\ \ k >= 0 \tag{1}$$

$$V_k(s) = \max_a Q_k(s,a),\ \ for\ \ k > 0 \tag{2}$$

In layman's terms, the value of a state transition is the sum of the instantaneous reward of the transition and the probability of that transition, multiplied by the discounted value of being in the new state. The value of being in a state is the value of the taking the action that maximizes the value of state transition from the current state. This mutually recursive relation, converges when $\lambda < 1$.

Markov Game Abstraction

The Markov abstraction of games, introduced by Littman[2] (shown in Figure 4) in 1994, allows us to define learning algorithms on games that have a certain structure. This structure is defined as the quadruple

$$\{S, A, T : P(s'|s,a), R : R(s,a)\},$$

where S is the set of states, A is the set of joint actions, T is the transition function with P being a discrete probability distribution over all states conditioned on the joint action and prior state (s, a), and R is the reward function based on the prior state and action (s, a). Of particular note is the set of joint actions A. In a turn-taking game, the joint action space will contain a no-op action for the player whose turn it is not; however, in a simultaneous-action game, the joint action space will contain the product of the action spaces of each of the players who can move simultaneously: (i.e. $\{A_i\ \ X\ \ A_{-i}\}$ for a two player game). Here, I deviate in notation from Littman, who specified the actions of both player's separately rather than creating a joint action space.

Figure 4. Richard Bellman (left) and Michael Littman (right). Richard Bellman from The RAND Cooperation was instrumental in creating algorithms to solve Markov Decision Processes (MDP)s. Michael Littman from Brown University used Richard Bellmans' ideas to create a way of modeling games after MDPs to apply convergent iterative algorithms to help agents learn rational policies.

The reason Markov games are so powerful is that they allow us to utilize tools that exist for other Markov processes for games, such as the method of value iteration describe previously. You'll notice that the main difference in definitions between a Markov Game and a MDP is that the agent doesn't control the whole action space. In fact, this formulation is a strict generalization of MDP's when the other player doesn't have any actions, and matrix games when there is only one state. By knowing the instantaneous values of any state transition, you can then determine what other states are desirable to be in, and that will lead to you the optimal policy of which actions to choose in which states.

Using the technique of value iteration, we can rephrase the value iteration equations to account for two players in a maximin game as follows:

$$Q_{k+1}(s,a,a_{-i}) = R(s,a,a_{-i}) + \gamma * \sum_{s'} T(s',s,a,a_{-i}) * V_k(s'),\ for\ k\ >= 0 \quad (3)$$

$$V_k(s) = \max_{\pi \in P(A)} \min_{a_{-i} \in A_{-i}} \sum_{a \in A} Q_k(s,a,a_{-i}),\ for\ k > 0 \quad (4)$$

Note that the only changes are that the other agent's actions are taken into account and that the maximum is over all probability distributions of actions that could be taken. A specific probability distribution for which action to take in each state, denoted as π, is found that maximizes the equation for V.

Desired Properties of Learning Agents

There are many desired properties of learning agents. Some are viewed as more important than others, and for some it is questioned whether they are really even important at all. The following section will explain in depth a few common properties discussed in the literature.

Convergence Convergence is the property that the agent should learn, or converge, to a good solution or strategy quickly, and not be stuck in a loop of changing between sub-optimal policies. In other words, the learning process should produce a sequence of policies whose optimality increases monotonically. As far as convergence is concerned, converging faster is more desirable, as long as the convergence still is monotonic.

Convergence is one property that is frequently looked at in developing agents, because a non-convergent algorithm could get stuck in a local minimum and not make any progress. However, it is a property that is also debated, since an algorithm that converges struggles to adapt to fast-changing strategies of an opponent or handling many different kinds of agents. Most papers present empirical tests for convergence in self-play and thus make it a weaker property than if theoretically proven.

Figure 5. Bowling is an important contributor in the field of game-theoretic agents, introducing agents that address convergence and regret.

Rationality Rationality is when the agent learns a best response to the other player, assuming the other player's policy does not change (i.e., it is stationary).

Optimality Foremost among desired properties of learning agents is the idea of optimality. Optimality in learning is often defined as being able to learn a strategy that achieves a Pareto optimal score, i.e., a score that is not Pareto dominated.

Sometimes this definition of optimality can be relaxed to also include Nash equilibria. As discussed, when talking about the Folk Theorem, any score above the minimax values can be sustained as a Nash equilibrium in a repeated game.

Optimality and rationality are closely related, so sometimes it is hard to draw the boundary between the two. The main difference is that rationality only strives to achieve best response results for the one player. In some papers, this would be called optimality. However, in others, optimality looks at the total payoffs for both players, and tries to maximize the social welfare by obtaining results that are Pareto-optimal or Pareto-optimal within some margin ϵ.

Regret Another interesting property is the concept of regret, introduced by Sergiu Hart and Andreu Mas-Colell[10] in the year 2000. Regret can be measured in many ways, but the most general definition is determining if there was a different strategy that you could have played than the one you did last round that would have been

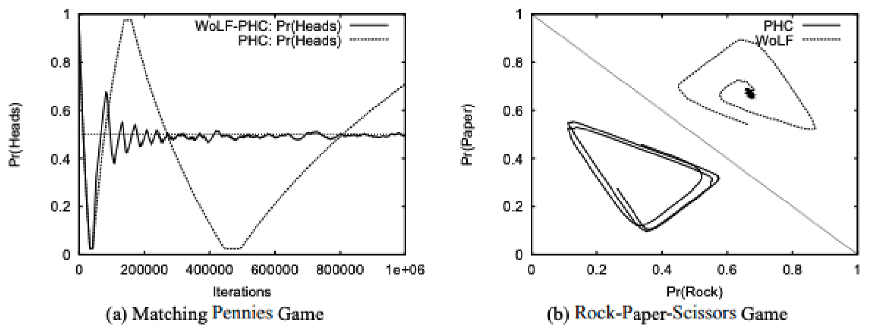


Figure 6. Graphs from Bowling's WoLF paper [8] showing the convergence of WoLF in a) Matching Pennies and b) Rock, Paper, Scissors. WoLF is denoted WoLF-PHC in the graph, and a similar algorithm that doesn't use the Win or Lose Fast principle is just called PHC (Policy Hill-Climbing).

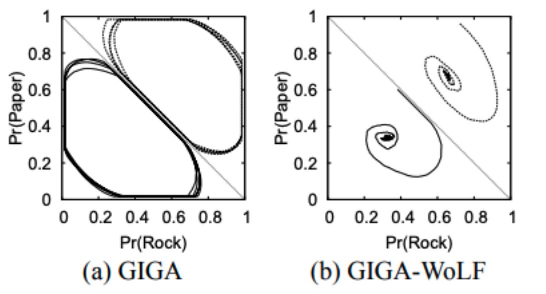


Figure 7. Graphs from Bowling's GIGA-WoLF paper [9] showing the convergence of a) the GIGA agent which is a agent based on no-regret and b) the convergence of GIGA-WoLF which keeps the no-regret property from the former, while also addressing convergence via the Win or Lose Fast principle.

better given the entire history of play. Regret would be the amount you lost by playing that previous strategy rather than the better strategy.

Minimax-Q

Value iteration is a great way to calculate the Q-values for different state action pairs, and it has the property of convergence. However, it is not necessarily practical or computationally tractable to iterate until convergence, especially when the state space is large. Littman, in his paper introducing the Markov game format[2] proposes to use the Q-learning approach to solve the same problem. In this approach, the same equations are used as the value iteration equations from above, but the updates are made asynchronously whenever an agent tests an action in a state. The update is performed only for the state action pair that is attempted. Since the transition probability is related to the actual observed outcome, the formulation is strictly equivalent to value iteration and will obtain the same Q-values in the limit. There are two additional parameters that this introduces, including a learning rate and decay factor for the learning rate, and an exploration versus exploit term which determines whether to take an action at random or use the learned model. It differs from the traditional Q-learning algorithm in the fact that it takes the minimax assumption into account, and therefore does much better about hedging its bets, instead of assuming the other agent won't change it's strategy.

WoLF

A significant contribution after the formulation of the minimax-Q learning agent is the WoLF (Win or Learn Fast) agent developed at Carnegie Mellon University by Michael Bowling and Manuela Veloso[8] in 2001. It strives to achieve not only convergence, but also rationality. This is stronger than just achieving the minimax solution. It builds off the Q-Learning formulation by keeping track of the policy rather than just the Q-values. That is, it keeps track of the probability distribution over actions to select, and adjusts that probability distribution based off the learning. In addition, it keeps track of the average policy and changes the policy more quickly when it is losing, compared to when it wins. This is achieved by having two different learning rates. It compares the average policy to the expected value of playing the current policy in the current state, and based on whether the current policy out-performs the average policy it will choose between the two learning rates. The average policy is a running average of all prior policies. Shown in Figure 6 we see that the algorithm converges quickly, unlike other agents that can get stuck in a loop by always switching strategies.

GIGA-WoLF

GIGA WoLF[9] developed in 2005 by Bowling is a variant of WoLF that is an example of an algorithm that satisfies the property of no-regret. In other words, it plays better than any static strategy could. It also has the property of convergence, converging in self-play to a static strategy as shown in Figure 7. GIGA-WoLF works by essentially keeping two policies that get updated independently. It improves on an earlier agent called GIGA that has the no-regret property. It addresses convergence through the Win or Lose Fast principle by updating the policy at different speeds depending on the outcome. At the same time, it ensures that it is improving with respect to regret by incorporating another independently-updated policy that keeps track of the regret.

Conclusion

Agents will always be a part of the society that we live in. Learning agents can be powerful, and agents that learn based off of game theory are even more powerful. Though the Markov game framework introduced by Littman is not the only way to think about games, it encompasses a large variety of games that can be applied to real-life problems. This simple abstraction leads to agents that can reason about how to maximize their own rewards or to jointly optimize the rewards to each individual agent. Understanding the learning characteristics and payoff characteristics of each learning agent allows us to choose an appropriate learning agent for each real-life application. Each agent has tradeoffs in their convergence, rationality, optimality, or regret characteristics based on the opponent they are facing. For a continually-learning agent that needs to adapt to changing circumstances, the convergence property might be less useful than an agent that acts optimally most of the time. A regret-based heuristic might be best applied when the other agent plays a largely static strategy or changes its strategy less frequently than the agent does. It is important to analyze the contexts in which an agent does well, which most papers highlight. Equally important, however, is understanding the faults of each agent including which assumptions it makes, or which properties it prioritizes over others. Every agent has a weakness, just like we do. A game theoretic agent might be able to manipulate your actions on the internet, but you have the power to subvert the agent by understanding how it works and changing tactics accordingly.

REFERENCES

1. J. v. Neumann, “Zur theorie der gesellschaftsspiele,” *Mathematische annalen*, vol. 100, no. 1, pp. 295–320, 1928.
2. M. L. Littman, “Markov games as a framework for multi-agent reinforcement learning,” in *Machine learning proceedings 1994*, pp. 157–163, Elsevier, 1994.
3. V. Pareto, “Manuale di economia politica,” 1906.
4. V. Pareto, “Manual of political economy,” 1971.
5. J. F. Nash *et al.*, “Equilibrium points in n-person games,” *Proceedings of the national academy of sciences*, vol. 36, no. 1, pp. 48–49, 1950.
6. J. W. Friedman, “A non-cooperative equilibrium for supergames,” *The Review of Economic Studies*, vol. 38, no. 1, pp. 1–12, 1971.
7. R. Bellman, “A markovian decision process,” *Indiana Univ. Math. J.*, vol. 6, pp. 679–684, 1957.
8. M. Bowling and M. Veloso, “Rational and convergent learning in stochastic games,” in *International joint conference on artificial intelligence*, vol. 17, pp. 1021–1026, Lawrence Erlbaum Associates Ltd, 2001.
9. M. Bowling, “Convergence and no-regret in multiagent learning,” in *Advances in neural information processing systems*, pp. 209–216, 2005.
10. S. Hart and A. Mas-Colell, “A simple adaptive procedure leading to correlated equilibrium,” *Econometrica*, vol. 68, no. 5, pp. 1127–1150, 2000.

Tim Whiting is in the Computer Science Masters and PhD program at Brigham Young University. His research involves game theory and learning agents embodied in robotic systems which interact and cooperate with humans in the real world through strategic communication.

The Development of Sorting Algorithms

J. D. Hutchings
Computer Science Department
College of Physical and Mathematical Sciences
Brigham Young University, Provo, Utah

***Abstract*—Sorting is a common problem that computer programs frequently encounter. Since the middle of the twentieth century, a variety of sorting algorithms have been developed, documented, analyzed, and modified to solve this problem. Each algorithm has performance strengths and weaknesses. To better analyze algorithms, a notation called "Big O" has been developed. Big O is useful for conveying how the performance of a sorting algorithm will be affected by the size of the list to be sorted. Big O can be expressed in terms of a variety of factors, such as memory requirements, steps, and operations. Having such knowledge is useful for determining how an algorithm may be used or modified to achieve better performance. Data structures and traversal methods can impact an algorithm's performance. Creative modifications can yield a high probability of significantly improved performance. Algorithms can be combined to produce hybrids that mitigate each algorithm's performance drawbacks and utilize each algorithm's performance advantages.**

■ **SORTING GROUPS OF ITEMS** is a process familiar to many. Students are sorted by last name for roll call; people are sorted by age to determine who goes first in a game; and shopping items can be sorted by a variety of preferences, such as price, customer ratings, and brand. Sorting can be tedious and time consuming, even when there are relatively few items to sort. Fortunately, sorting is a task well suited to computers.

Many sorting algorithms have been developed and studied in depth. Each method has certain advantages and disadvantages. Although sorting may seem like a simple task, requirements of the various sorting algorithms and limitations of computer hardware necessitate careful analysis and thoughtful selection of suitable algorithms. A primary consideration is computational complexity, which deals with the resources required by a given algorithm.

Space and time are two of the most common resources that are accounted for in computational complexity. Space refers to the amount of memory that is required. Time refers to the number of steps that are performed. Other things that can be accounted for when analyzing the computational complexity of sorting algorithms are the number of swaps and comparisons that are performed.

Big O notation, which was first introduced by Paul Bachmann[1], is a useful notation for expressing computational complexity. It gives an approximation of how the performance of an algorithm is affected by the size of a problem. The size of a sorting problem is the size of the list that is to be sorted. A sorting algorithm's performance can be measured in terms of previously mentioned things, such as space, time, swaps, and comparisons.

Some of the more well-established comparison sorting algorithms can be grouped into simple and efficient sorting algorithms. Simple sorting algorithms tend to perform poorly on large lists when compared to their efficient sorting algorithm competitors, but they sometimes have other desirable traits. For example, a sorting algorithm is called ”online” if it can successfully sort a list even after an item has been appended to the list while the list was being sorted.

Although there are standard methods of implementing sorting algorithms, creative liberty can be taken to increase the probability of high performance. Sorting algorithms can also be combined to form hybrid sorting algorithms, allowing judicious application of each sorting algorithm in order to take advantage of its performance benefits while mitigating or eliminating the performance drawbacks of another.

Big O Notation

Donald Knuth made significant contributions to the computer science discipline by laying out many algorithms and using Big O notation.[2] Big O notation is written as $O(f(n))$, and Knuth's definition is ”There are positive constants M and n_0 such that the number x_n represented by $(O(f(n))$ satisfies the condition $|x_n| \leq M|f(n)|$, for all integers $n \geq n_0$.” In practice, Big O is used as an approximate measure of how an algorithm's performance is affected by the size of the problem it is solving. Relatively insignificant details are often left out of the Big O representation. For example, if a function were to perform one step per value in n and an additional 5 steps, the Big O representation would be commonly expressed as $O(n)$ rather than $O(n+5)$. This is because as the value n grows, the 5 remains constant and becomes relatively insignificant. Some examples of Big O are shown in Figure 1. Notice that constant factors are omitted, sums of the same function are reduced to $f(n)$, and each distinct function is expressed independently of other functions.

$$f(n) = O(f(n)), \tag{5}$$
$$c \cdot O(f(n)) = O(f(n)), \qquad \text{if } c \text{ is a constant}, \tag{6}$$
$$O(f(n)) + O(f(n)) = O(f(n)), \tag{7}$$
$$O(O(f(n))) = O(f(n)), \tag{8}$$
$$O(f(n))O(g(n)) = O(f(n)g(n)), \tag{9}$$
$$O(f(n)g(n)) = f(n)O(g(n)). \tag{10}$$

Figure 1. Some common Big O operations[2]

To put Big O into a more relatable context, consider the task of washing dishes. The time complexity of washing dishes can be analyzed in terms of wash cycles. If the dishes are washed by hand, then one dish can be washed per wash cycle since a person can only wash one dish at a time. In Big O, the number of wash cycles would be expressed as $O(n)$, where n is the number of dishes that need to be washed. $O(n)$ is commonly called ”linear time complexity” since the number of steps to accomplish a task increases proportional to the size of the problem. The graph for this would look like a straight line with an upward slope. If the dishes are washed using a dishwasher, then many dishes can be washed per wash cycle. The number of wash cycles when using a dishwasher would be expressed in Big O as $O(1)$, assuming the number of dishes to be washed does not exceed the capacity of the dishwasher. This is commonly called ”constant time complexity” since the number of steps to accomplish a task remains the same regardless of the size of the problem. The graph for this would look like a flat line.

$O(1)$ time complexity appears to be better than $O(n)$ time complexity since many dishes can be washed simultaneously. If the dishwasher is capable of washing dozens or hundreds of dishes, then the dishwasher may very well be worth it. Suppose, however, that only a few dishes need to be washed. If a person can wash one dish fast enough, then enough time might be saved by hand washing those few dishes to be worth the effort. There are many other Big O expressions that are commonly used when analyzing algorithms. Figure 2 shows some Big O graphs.

Sorting Algorithms

Several fundamental sorting algorithms are discussed in the following sections. Each section will be dedicated to a single sorting algorithm and may include key points such as a description of the algorithm's various procedures, an analysis of its computational complexity, and other key aspects of the algorithm. The list in Figure 3 will be used to help describe the algorithms.

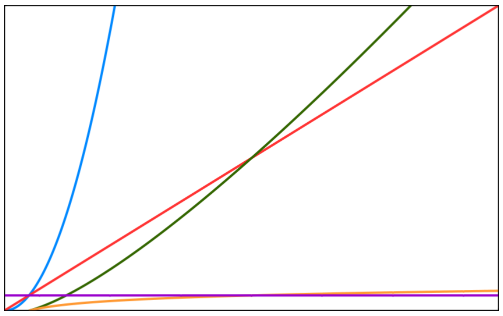


Figure 2. Graphs of common Big O expressions. As n grows, n^2 grows fastest, followed by $n\,log\,n$, then n, and $log\,n$. The constant function $f(n) = 1$ is not affected by the size of n. Also notice that for very small values of n, the faster growing graphs, which are less efficient on large values of n, may perform better than the other graphs.

The Big O representation of an algorithm's space complexity sometimes omits the space required to store the original list. In this paper, that space is included. For example, if the amount of additional memory that a sorting algorithm requires is constant regardless of the size of the list to be sorted, then $O(n)$ will represent the space complexity rather than $O(1)$.

4	2	3	7	6	9	8	10	1	5

Figure 3. The numbers 1 through 10 in an unsorted list

Insertion Sort

Insertion sort (see Figure 4) is a simple sorting algorithm. To begin, it makes two partitions of a list. One partition is sorted, and the other is not. At the beginning, the sorted partition contains the first item in the list and the unsorted partition contains the rest. It iterates over the unsorted partition, putting the next unsorted item into its place in the sorted partition. The unsorted item's new position is determined by comparing the unsorted item to each item in the sorted partition, beginning from the end, until an item that is less than or equal to the unsorted item is found. The unsorted item is placed behind that sorted item, making the unsorted item now a sorted item. Of course, this might not be the final position for that item since it would be pushed back once per remaining unsorted item that is less than it.

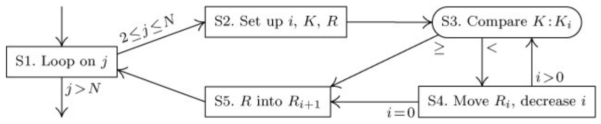


Figure 4. Insertion sort, as shown by Donald Knuth in *The Art of Computer Programming: Volume 3: Sorting and Searching*[3]

Like other simple sorting algorithms, insertion sort's time complexity would be represented in Big O notation as $O(n^2)$. This is due to the nested looping structure. There is one loop, called the outer loop, to iterate over the unsorted partition of the list, and another loop, called the inner loop, to iterate over the sorted partition of the list. The inner loop is performed for each iteration of the outer loop, so the time complexity will be determined by multiplying the time complexity of the inner loop by the time complexity of the outer loop. The outer loop iterates approximately n times, and the inner loop potentially iterates over all the items in the sorted partition of size $n - 1$ on the last iteration of the outer loop. The result is $n(n - 1)$. Big O notation ignores the constant -1 since n is higher order, so it would be written as $O(n^2)$.

Insertion sort is done in-place. It only requires the amount of space to store the list that is being sorted and the space required to support the looping structures, comparisons, and swaps. $O(n)$ space is required to store the list. $O(1)$ space is required for the supporting operations. The space required for the supporting operations is ignored because it is lower order than the space required to store the list, so the overall space complexity is $O(n)$.

Insertion sort is a stable sort. This means that the order of equivalent items does not change. Suppose item a and b are equal and that a came before item b prior to sorting the list. After sorting the list, item a will still be before item b. Insertion sort is also online, meaning that an item can be appended to the end of the list while the list is being sorted and the algorithm will still sort it properly. It is also adaptive, meaning the original order of items in the list affects its performance. Consider a list where the first half of its items happened to be sorted before being provided to insertion sort. Insertion sort will compare each of those items to one item (the item immediately before it), perform no swaps, and move on to the next item. This is far better than its performance on items that are sorted in reverse; each unsorted item will be compared to and swapped with every sorted item. A presorted list is the best-case scenario, resulting in $O(n)$ comparisons and no swaps.

Data structures available to computers allow for improvements to be made to insertion sort. Linked lists allow for improved swap performance, and binary search allows for quickly finding the new location for an item.

Items in a linked list are not necessarily stored next to each other within a computer's memory, as the items in Figure 3 appear to be. A visual representation of a linked list is shown in Figure 5. The arrows represent links that maintain the order in which items appear in the list.

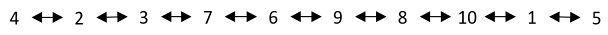


Figure 5. A linked list built from the list in Figure 3

A disadvantage to the linked list data structure is the speed of accessing an item in the list. Suppose, for example, that a program will retrieve the item in the fourth position in the list. If the items are stored next to each other in a computer's memory, then the program can simply leap from the first item in the list straight to the fourth item. Accessing any given item in such a list has $O(1)$ time complexity since the item can be found in a single step regardless of its position within the list. To retrieve the fourth item in a linked list, however, is not so simple; the program begins at the first item in the list, uses the first item's link to find the second item, then uses the second item's link to find the third item, and then uses the third item's link to find the fourth item. Accessing any given item in a linked list has $O(n)$ time complexity since it must traverse some fraction of the total number of links to find the item. Fortunately, this inefficiency does not affect insertion sort, since both of insertion sort's loops iterate over the items in a list one at a time, anyway.

An advantage of the linked list data structure is the ease of moving items. Suppose the sixth item in a list is to be placed between the first and second items. A list with items stored next to each other must shift the second through the fifth items over one position, then put the item that was in the sixth position into the second position. This operation has $O(n)$ time complexity, since the number of items to be shifted is some fraction of the total number of items. A linked list, on the other hand, simply breaks and reforms the links for the sixth and second items. This operation has $O(1)$ time complexity, since there is a constant number of links to break and reform. This is very advantageous for insertion sort since the number of swaps drops by a factor of n, reducing the swap complexity from $O(n^2)$ to $O(n)$.

The number of comparisons can be reduced from $O(n^2)$ to $O(n\,log\,n)$ by using a binary search. A binary search is performed on a sorted list by repeatedly comparing an unsorted item to the item that is in the middle of the list or, if the list has an odd number of items, to an item that is as close to the middle as can be. If the unsorted item is larger than the approximate middle item, then binary search continues the same method of finding and comparing with the items that come after the approximate middle item. If the unsorted item is smaller, then binary search continues with the items that come before the approximate middle item. Otherwise, the items are equal, and the unsorted item can be inserted after the approximate middle item. Binary search is useful for lists that are stored with items side by side in memory because it can take advantage of the $O(1)$ time complexity of accessing the middle item. Unfortunately, it is not as useful with linked lists because the time complexity of accessing the middle item in a linked list is $O(n)$.

In 2006, Bender, Farach-Colton, and Mosteiro found that insertion sort could be modified to achieve a high probability of $O(n\,log\,n)$.[4] They called this modification ”Gapped Insertion Sort” or ”Library Sort.” The high probability of such improved performance is achieved by leaving gaps between items during the insertion process, which results in fewer swaps.

Heapsort

Heapsort is an efficient sorting algorithm that was invented by J. W. J. Williams.[5] It accomplishes its tasks in two parts. First, it partitions the list into a heap data structure, specifically a max-heap, and a sorted partition. At the beginning, the heap contains all the items and the sorted partition is empty. The items in the heap are stored in front of the items in the sorted partition. Once the two partitions have been created, heapsort moves one item at a time from the heap to the sorted partition.

A heap is composed of nodes (items in a list). Each node can be a parent node, a child node, or both a parent and a child node. Each parent node can have at most two child nodes. Each child node is positioned relative to its parent node. A max-heap requires that each parent node be greater than or equal to its child nodes. Take, for example, the list in Figure 3. Turning that list into a max-heap would result in the list in Figure 6. Notice that the root node (the first item) in the heap is the largest item. The root node is a parent of the second and third nodes, the second node is a parent of the fourth and fifth nodes, the third node is a parent of the sixth and seventh nodes, the fourth node is a parent of the eighth and ninth nodes, and the fifth node is a parent of the tenth node. The position of child nodes relative to their parent nodes is calculated by $2p + 1$ for the first child node and $2(p+1)$ for the second child node, where p is the position of the parent node and the position of the root node is 0. Notice that the nodes in the heap are grouped by generation. A convenient visualization of a heap is the tree in Figure 7, where the numbers are nodes, and the parent-child relations are indicated by lines from one node to another.

10	7	9	4	6	3	8	2	1	5

Figure 6. A max-heap built from the list in Figure 3

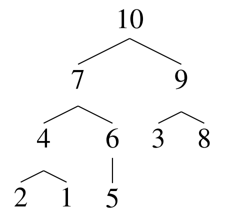


Figure 7. A tree representation of the max-heap shown in Figure 6

Ryan Hayward and Colin McDiarmid[6] found that Floyd's[7] method of building a heap was more efficient than the original method proposed by Williams, so Floyd's method was used to build the heap in Figure 6 from the list in Figure 3. First, the list was assumed to be a heap that did not satisfy the max-heap requirement that each parent node be greater than or equal to its child nodes. Then, each parent node in the heap, beginning from the last parent node and ending with the root node, was sifted down the tree until both of its child nodes were less than or equal to it. Sifting parent nodes down the tree is accomplished by swapping the parent node with the larger of its child nodes, if it is less than at least one of its child nodes. Then, it is swapped in the same way with its new child nodes until it has reached the bottom of the tree or both of its child nodes are greater than or equal to it. Building a max-heap using this method has $O(n)$ time complexity.

Once the max-heap has been built, items are removed from the heap to the sorted partition of the list. This is accomplished by repeatedly performing two sets of operations until there are no items left in the heap. First, the root node of the heap is moved to the front of the sorted partition. Then, the last node in the heap is moved to the front of the heap, becoming the heap's new root node. The new root node is then sifted down following the same procedure that was used when sifting down parent nodes to build the heap. The time complexity of this procedure is the time complexity of moving the items from the heap to the sorted partition multiplied by the time complexity of sifting the last node in the heap down from the root node. Moving the items has $O(n)$ time complexity, since n items are moved. Due to the heap's tree structure, the most items that could be swapped with the root node as it is sifted down is approximately $log_2(n)$, so sifting the root node has $O(log\, n)$ time complexity. This yields a time complexity of $O(n\, log\, n)$ to move items from a heap to a sorted list.

The overall time complexity of heapsort is determined by adding the time complexity of building a heap to the time complexity of moving items from a heap to a sorted list. $O(n) + O(n\, log\, n)$ yields $O(n\, log\, n)$. $O(n)$ is left out because it is lower order than $O(n\, log\, n)$.

Merge Sort

Merge sort (see Figure 8) is a well-known efficient sorting algorithm. Rather than swapping items in place, it recursively splits a list in two until there is no more than one item in each list. Then it merges these newly created lists together. A list of size one is already sorted, and the merging algorithm ensures that the resulting list is also sorted. The first item in each list is compared, and the smaller of the two is appended to the end of a results list. This continues until one of the lists is empty, at which point the remaining items are all appended, maintaining their current order, to the results list.

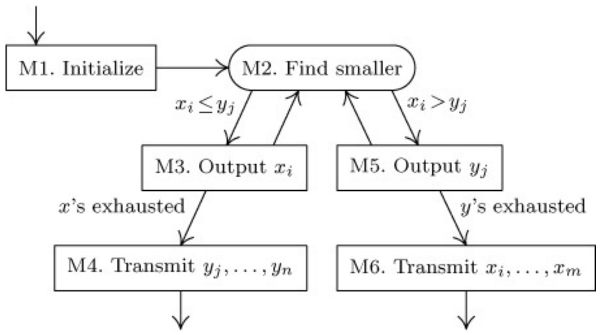


Figure 8. The merging algorithm of merge sort, as shown by Donald Knuth in *The Art of Computer Programming: Volume 3: Sorting and Searching*[3]. Variables i and j are initialized to the index of the first item in their lists, and variables n and m are the number of items in the lists that are being merged.

The time complexity for merge sort is $O(n \, log \, n)$. This can be seen by expressing the running time of merge sort as a recurrence relation, then by applying the master theorem.[8] First, express the running time of merge sort as the function $T(n)$, where n is the number of items in a list. Its recurrence relation would be expressed as

$$T(n) = 2T(n/2) + n$$

$2T(n/2)$ because the list is split into two lists of approximately equal size, both of which are then passed through merge sort again. $+n$ because each merge takes n assignments and no more than n comparisons. The master theorem shows that recurrence relations of the form

$$T(n) = kT(n/c) + f(n)$$

can be expressed in Big O terms as $O(n \, log \, n)$.

The space complexity for merge sort is very straightforward. It requires enough space to store the original list, and an additional amount of space to store the results list while merging two lists together. The largest results list is the same size as the original list, so merge sort requires $O(n)$ space.

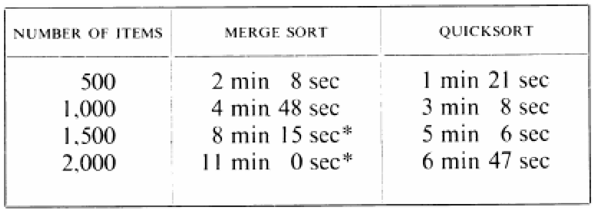
There are many other efficient sorting algorithms and variations of merge sort that are interesting to compare with this description of merge sort, such as heapsort, which has already been discussed, and quick sort, which is discussed in the next section. Factors such as space limitations and order stability can affect which is best for specific scenarios. Merge sort is a stable sort, whereas quicksort may not be.

Quicksort

Quicksort is an efficient sorting algorithm that was invented by Sir Charles Antony Richard Hoare in 1959.[9] The algorithm recursively creates two partitions of a list based around pivot values. Items in the list are swapped during the partitioning processes. Once all the partitions have been created, the algorithm is finished.

Although Hoare's partitioning algorithm is not the only partitioning algorithm that has been developed for quicksort, it is used here.[10] First, a pivot value is chosen. Second, coming from the beginning and going to the end of the list, the first location where an item is greater than the pivot is found. Third, coming from the end and going to the beginning of the list, the first location where an item is less than the pivot is found. The fourth step is conditional upon the locations found in steps two and three. If the location found in step two comes before the location found in step three, then the items in those two locations are swapped. Then, steps two, three, and four are repeated, except the searches in steps two and three begin from where they left off rather than from the beginning and end of the list. Once the locations found in steps two and three cross, the list is partitioned between the two locations. The partitioning process is then run on those two new partitions. If a partition contains one or fewer items, then it is not partitioned further. If a partition contains a small number of items, then it may be preferable to sort it using some other algorithm; although, doing so would make the algorithm a hybrid of quicksort and whichever algorithm is used for small lists.

The choice of pivot is crucial for quicksort to be efficient. Without proper pivot selection, it is even possible for the algorithm to never reach completion. Suppose that the value 20 is chosen as the pivot for the unsorted list in Figure 3. The partitioning algorithm would produce one list that contains all the items from the original and another that is empty. No sorting would have been accomplished and the full original list would need to be partitioned again. If pivot values are chosen that are less than or greater than all the items in the list, then the algorithm would never reach completion. A good implementation of quicksort produces excellent results. Figure 9 shows results of merge sort and quicksort on lists of four different sizes.



NUMBER OF ITEMS	MERGE SORT	QUICKSORT
500	2 min 8 sec	1 min 21 sec
1,000	4 min 48 sec	3 min 8 sec
1,500	8 min 15 sec*	5 min 6 sec
2,000	11 min 0 sec*	6 min 47 sec

Figure 9. Hoare's comparison of the results of merge sort and quicksort[10]

Timsort, A Hybrid Sorting Algorithm

Timsort is a hybrid sorting algorithm developed by Tim Peters for the Python programming language.[11][12][13] It combines insertion sort and merge sort and was designed to take advantage of lists that contain runs of sorted items. A run can be ascending, where each item is less than or equal to the next, or it can be descending, where each item is greater than the next. Descending runs are reversed to create ascending runs. Since timsort is a stable sorting algorithm, descending runs cannot have items that are equal; otherwise, when a descending run is reversed, the order of equal items would also be reversed.

One of the main differences between timsort and merge sort is the method of creating sorted sequences that will be merged. Merge sort recursively divides a list into smaller lists and merges them together. Timsort uses existing sorted runs and creates other sorted runs using insertion sort with binary search.

To create partitions of sorted runs, timsort first determines a minimum run length. Next, it scans through the list, looking for sorted runs that are already present and are at least as long as the minimum run length. If a run is shorter than the minimum run length, then it is extended by applying a binary insertion sort to all of the items from the start of the run to the minimum run length past the start of the run.

A variety of factors affect how timsort determines the minimum run length. Two considerations when selecting the minimum run length are the special cases when the size of the list is less than sixty-four and when the size of the list is a power of 2. When the size of the list is less than sixty-four, the minimum run length is the size of the list, meaning that insertion sort is used to sort the entire list without performing any merges. When the size of the list is a power of 2, using a minimum run length that is equal to a power of 2 is preferred because it results in merging lists of equal size.

Peters points out that a minimum run length should be avoided if it satisfies the equation

$$q, r = divmod(N, minrun)$$

when q is a power of 2 and $r > 0$. $divmod$ is a Python function[1], q is the quotient, r is the remainder, $minrun$ is the minimum run length, and N is the size of the list.

The locations and lengths of runs that timsort finds and creates are stored on a stack. A stack is a data structure that follows the last in, first out inventory methodology; that is, items are always appended and removed at the top of the stack. Two of the three topmost runs are merged when either of two criteria are met. First, if the sum of the lengths of the two topmost runs is greater than or equal to the length of the third run, then the second run is merged with the smaller of the first and third runs. If the first and third runs are equal in length, then the first and second runs are used for the merge. Second, if the length of the

[1]https://www.programiz.com/python-programming/methods/built-in/divmod

second topmost run is less than the length of the topmost run, then those two runs are merged.

Timsort and merge sort's merge processes are distinct. Timsort's merge process uses a temporary work list that is no larger than the smaller of the two runs being merged. Items of one run are stored in the temporary work list to make room for repositioning items in the original list. Then, items are compared and moved in a similar way to merge sort. Two items, one from each run, are compared and one is moved, if necessary. This is repeated until one run is empty. If the temporary work list has any items left over, then they are inserted into the remaining unfilled locations in the original list. The merge is complete when the temporary work list is empty.

Peters uses a hybrid of samplesort[14] to compare the results of timsort's performance. Timsort proved to be clearly superior with lists that already contained many runs, which is what it was designed to do. Timsort exemplifies thoughtful consideration of real-world circumstances.

CONCLUSION

Problems that have existed since before the age of computers require careful and thoughtful analysis. Although a process may appear to be simple, there are often hidden details that affect its practicality. Big O notation facilitates simple communication of key factors that affect a given algorithm's performance, which provides direction when deciding how to use or improve the algorithm. Although an algorithm may appear to be totally inferior to another, continually analyzing it and researching new ways to implement and utilize it is still valuable; it may be useful as a hybrid, or new information might reveal more efficient implementations. The development of sorting algorithms has benefited from such continual analysis and experimentation. The performance of insertion sort can be improved via linked lists, binary search, and spacing. The hybrid sorting algorithm, timsort, achieves phenomenal performance on a variety of data sets by making use of insertion sort's good performance on small, nearly sorted lists, while utilizing a highly efficient merge process. It will be exciting to observe the continued development of sorting algorithms as new and creative methods are explored.

REFERENCES

1. P. Bachmann, *Analytische Zahlentheorie.* Leipzig: Teubner, 1894.
2. D. Knuth, *The Art of Computer Programming: Volume 1: Fundamental Algorithms.* Addison Wesley Longman, 3 ed., 1997.
3. D. Knuth, *The Art of Computer Programming: Volume 3: Sorting and Searching.* Addison-Wesley Professional, 2 ed., 1998.
4. M. Bender, M. Farach-Colton, and M. Mosteiro, "Insertion sort is o(n log n)," *Theory of Computing Systems*, vol. 39, pp. 391–397, 2006.
5. G. E. Forsythe, "Algorithms," *Commun. ACM*, vol. 7, p. 347–349, June 1964.
6. R. Hayward and C. McDiarmid, "Average case analysis of heap building by repeated insertion," *Journal of Algorithms*, vol. 12, no. 1, pp. 126 – 153, 1991.
7. R. W. Floyd, "Algorithm 245: Treesort," *Commun. ACM*, vol. 7, no. 12, p. 701, 1964.
8. J. L. Bentley, D. Haken, and J. B. Saxe, "A general method for solving divide-and-conquer recurrences," *Science in Context*, vol. 14, no. 4, pp. 591–614, 2001.
9. "Sir antony hoare - chm," 2006. https://computerhistory.org/profile/sir-antony-hoare/, accessed: 11.23.2020.
10. C. A. R. Hoare, "Quicksort," *The Computer Journal*, vol. 5, pp. 10–16, 01 1962.
11. T. Peters, "listsort," 2002. https://hg.python.org/cpython/file/d971b8703508/Objects/listsort.txt, accessed: 11.07.2020.
12. T. Peters, "listsort." https://svn.python.org/projects/python/trunk/Objects/listsort.txt, accessed: 11.07.2020.
13. B. Peterson, "listobject.c," 2016. https://hg.python.org/cpython/file/ec537f9f468f/Objects/listobject.c, accessed: 11.08.2020.
14. W. D. Frazer and A. C. McKellar, "Samplesort: A sampling approach to minimal storage tree sorting," *J. ACM*, vol. 17, p. 496–507, July 1970.

Jeffrey Hutchings is a Master's Student in Computer Science (CS) at Brigham Young University (BYU). He received a B.S. degree in Wildlife and Wildlands Conservation with a minor in CS from BYU in 2016. For his Master's project, he built an automated speech therapy website that prompts a user to speak a sentence, runs speech-to-text on the user's audio input, and reports how accurately the speech matches the prompt. Email: jdanhutch@gmail.com.

History and Development of Internet

Tarun Kumar Yadav
Brigham Young University

***Abstract*—Throughout human history, communication processes are evolved from physical delivery of messages, which could take up to months, to message delivery through the Internet, which takes less than a few seconds to reach any part of the world. Today, the Internet does more than just connect computers. It connects people, lives, stories, and businesses. It is a source of information, a social platform, and a business network. The Internet has diverse uses depending on the needs of an individual or the setting. Regardless of the purpose, the importance of the Internet in our daily lives is unquestionable. In this paper, we discuss the history of the Internet and major milestones that helped build today's Internet. We start with the first communication devices, like the electrical telegraph. We then discuss the development of significant protocols and their importance.**

■ **The impact of the Internet** continues to increase in our lives, society, and culture. The very fact that we are taking online classes from around the world together requires a technological infrastructure that was designed, engineered, and built over the past seventy years. To function in an information-centric world, we need to understand the workings of network technology. We must understand the concepts/technology/protocols behind the Internet, how it was created, who created it, and how it works. Learning about the innovators who developed the Internet and Web technologies that we use today would help us understand the researchers' critical thinking who evolved a network of 4 computers (ARPANET) into a network of billions of connected computers.

The Internet is a global network of networks, a complex system that is evolved remarkably over the last 70 years. It started as a US defense project for the communication of armed forces over the connected network [1]. The ARPANET was the first connected network that connected military installations, third-party contractors, and a few universities in the US. By the mid-1970s, ARPANET had connected NORSAR, a US-Norwegian system designed to monitor seismic activity from earthquakes or nuclear blasts, over satellite, computers in London, and eventually, other parts of Europe. After decades of research and continuous evolution, the World Wide Web was developed with the help of a man named Tim Berners-Lee. The development of the HTTP protocol led to the development of browsers and with that, the popularity of computers skyrocketed. By 1992, more than a million computers were connected — only two years after HTTP was developed. Today the Internet is used directly or indirectly for almost every service like Internet Banking, Matrimonial Services, Online Shopping, Online Ticket Booking, Online Bill Payment, Data Sharing, and E-mail. Today's Internet is a complex system of complex interconnected networks and Security protocols that all work together. It uses TCP/IP as a standard network protocol, TLS as a standard network security protocol, DNS to retrieve IP addresses from domain names, FTP to retrieve a file, DHCP to assign an IP address to network endpoint, OSPF for routing, SMTP for E-mail communication, and UDP to transmit time-sensitive data. In the following sections, we discuss the development of these protocols and how they helped shape the current Internet.

Communication before Internet

Physical message delivery

The early period of human history required the physical delivery of messages to communicate. People would travel to communicate messages or shouting had worked for a close neighborhood. In the first century BCE, people started using pigeons to send messages [2]. Pigeons were effective as messengers due to their natural homing abilities. The pigeons were transported to a destination in cages, where they would be attached with messages, then the pigeon would naturally fly back to its home where the recipient could read the message. One interesting fact is Pigeons were used to deliver the results of the first ancient Olympics in 776 BCE [3].

The pigeons used to fly with an average speed of 50mph but couldn't travel for long distances. To communicate for long distances, people started using horses [4]. The couriers on horseback could travel for 2,000 miles in a week. In the 5th century BCE, the Persian king Darius the Great built the Royal Road to improve the couriers' path. But still, the message delivery could take up to a couple of weeks, depending on the roads (rocky mountains takes longer) and climate.

Telegraph: Grandpa of the Internet

In 1838 Samuel Morse, along with other scientists, developed the electric telegraph [5], which revolutionized long-distance point-to-point message communication. Morse sends the first message in 1844 from Washington, DC, to Baltimore, and by 1858 a telegraph line had been laid across the Atlantic Ocean from the US to Europe. The telegraph used Morse code to encode or decode the message in the electric signal. Morse code didn't survive the transition of Analog signals to Digital signals, but the binary coding (used in digital signals) was founded based on Morse code's principles of using simple and easily distinguishable signals to encode messages. The telegraph's main problem was that it used Morse code and was limited to sending and receiving one message at a time.

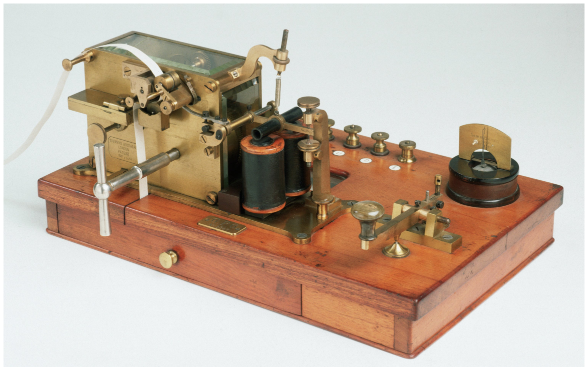


Figure 1. The Telegraph

Telephone

The mechanical telephones based on sound transmission through pipes and other physical media (like wires) have been known for centuries. During the 1870's Alexander Graham Bell and Elisha Gray independently invented the electrical telephone and registered the patents within hours of each other [6]. It led to a legal battle that was won by Bell. The telephone emerged from the making and successive improvements of the electrical telegraphs.

A circuit switch connects the output of one circuit to another's input, allowing information to be passed. In a circuit-switched network, end clients have access to the circuit's full bandwidth during the communication. Before 1891, circuits ran from point to point and were always connected over a physical facility such as a pair of wires. In the early days, this switching was done by human operators physically putting in plugs to connect lines.

In 1891, Almon Strowger revolutionized the entire telephone system by inventing an automatic telephone switching system [7] that allows people to dial each other directly, thereby eliminating any need for a telephone switchboard operator.

Development of the first Digital network

Like other evolution that happened in computer science, the Internet is the result of the constant development of new ideas. In this section, we describe the ideas that lead us towards today's Internet.

Figure 2. One of the original automatic circuit switches

Information theory

In 1948, Claude Shannon, who is considered the father of modern Information theory, defined a formal way of studying communication channels [8] in his Information Theory paper. This paper has been called "The Magna Carta of the Information Age"— meaning a founding document that inaugurated an era. It took Shannon 10 years to develop this idea that all communicating messages, irrelevant of the sender, receiver, and length of the message, can be represented in the form of bits. In this paper, he explains the compression and encoding of bits for information transmission with flawless accuracy. Shannon's paper has been considered a rare case where he finds a field and solves all the major problems in one stroke. This work is the reason for all the further progress in digital communication.

The first electronic digital computer was developed in 1942 by John Vincent Atanasoff and Clifford E. Berry, but the ideas for computer communications started in the early 1960s.

Packet switching

The credit for inventing packet-switching and laying the Internet's foundation can be given to three people: Leonard Kleinrock, Donald Davies, and Paul Baran.

In 1961, Leonard Kleinrock pioneers the packet-switching concept in his Massachusetts Institute of Technology (MIT) doctoral thesis about queueing theory[[9]]. There were a lot of computers at MIT during the time Leonard Kleinrock was doing his doctorate. He realized that eventually, all these computers need to communicate. And the mathematical descriptions of existing networks like telephone exchanges, where there is only a single node to node communication (through circuit switching), would be inadequate because there would be many nodes in the future. Therefore, he extended the mathematical description of *queuing theory* to work with large complex networks and described the mathematical description of *packet switching* [10], where each data stream is broken into discrete packets for transmission.

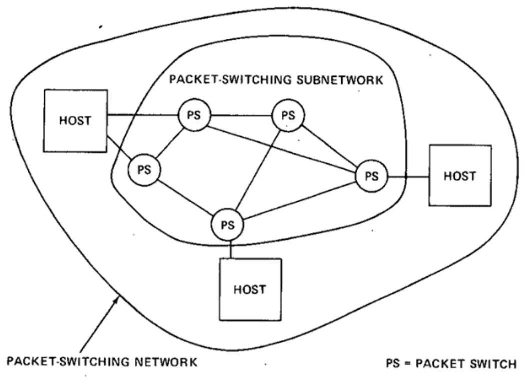


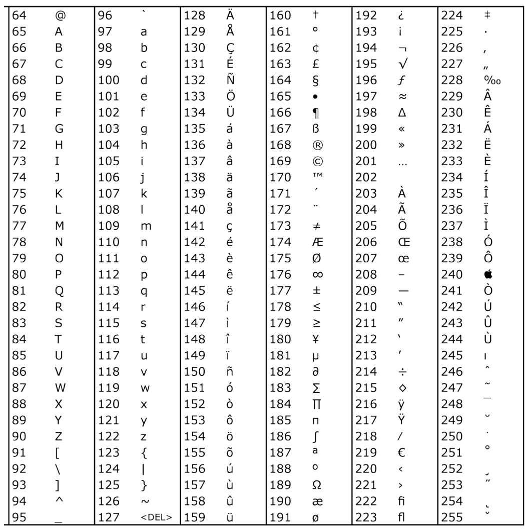
Fig. 1. Typical packet switching network.

Figure 3. Packet Switching

Between 1962 and 1964, Paul Baran invented the concept of *message blocks* [11]. His idea was to break the transmitted message into smaller packets, which could be sent to the destination independently. The packets are combined at the receiver end to reconstruct the original message. In fig 3, when any two hosts communicate using the packet switching approach, the multiple packets in between two hosts can take any route in between independently and reach the destination, where they are reconstructed. This approach of *message blocks* allows using dedicated lines for any number of circuits, unlike circuit switching, where one dedicated line allows one circuit. It increased transmission capacity and created a flexible, reliable, and robust communications network. Using this concept, he designed a decentralized and interconnected system of networks. These are the two essential properties that can even be seen on today's Internet. In 1965, Donald Davis in the UK (at NPL) independently developed the same concepts and named it ”packet switching” [12], which is the name that is used today for this concept.

Character encoding

A character encoding defines the interpretation of binary 0 and 1 sequences to characters. ASCII, the first universal standard for character encoding, was developed in 1963 by a joint industry-government committee. It allowed computers from different manufacturers to communicate. Fig4 is a simpler version of the ASCII table showing the mapping of only English letters to binary. ASCII was the most common character encoding on the World Wide Web until December 2007 [13], when UTF-8 encoding surpassed it; UTF-8 is backward compatible with ASCII.



64	@	96	`	128	Ä	160	†	192	¿	224	‡
65	A	97	a	129	Å	161	°	193	¡	225	·
66	B	98	b	130	Ç	162	¢	194	¬	226	‚
67	C	99	c	131	É	163	£	195	√	227	„
68	D	100	d	132	Ñ	164	§	196	ƒ	228	‰
69	E	101	e	133	Ö	165	•	197	≈	229	Â
70	F	102	f	134	Ü	166	¶	198	∆	230	Ê
71	G	103	g	135	á	167	ß	199	«	231	Á
72	H	104	h	136	à	168	®	200	»	232	Ë
73	I	105	i	137	â	169	©	201	…	233	È
74	J	106	j	138	ä	170	™	202		234	Í
75	K	107	k	139	ã	171	´	203	À	235	Î
76	L	108	l	140	å	172	¨	204	Ã	236	Ï
77	M	109	m	141	ç	173	≠	205	Õ	237	Ì
78	N	110	n	142	é	174	Æ	206	Œ	238	Ó
79	O	111	o	143	è	175	Ø	207	œ	239	Ô
80	P	112	p	144	ê	176	∞	208	–	240	
81	Q	113	q	145	ë	177	±	209	—	241	Ò
82	R	114	r	146	í	178	≤	210	“	242	Ú
83	S	115	s	147	ì	179	≥	211	”	243	Û
84	T	116	t	148	î	180	¥	212	‘	244	Ù
85	U	117	u	149	ï	181	µ	213	’	245	ı
86	V	118	v	150	ñ	182	∂	214	÷	246	ˆ
87	W	119	w	151	ó	183	∑	215	◊	247	˜
88	X	120	x	152	ò	184	∏	216	ÿ	248	¯
89	Y	121	y	153	ô	185	π	217	Ÿ	249	˘
90	Z	122	z	154	ö	186	∫	218	⁄	250	˙
91	[	123	{	155	õ	187	ª	219	€	251	˚
92	\	124	|	156	ú	188	º	220	‹	252	¸
93	]	125	}	157	ù	189	Ω	221	›	253	˝
94	^	126	~	158	û	190	æ	222	ﬁ	254	˛
95	_	127	<DEL>	159	ü	191	ø	223	ﬂ	255	ˇ

Figure 4. ASCII code table[*]

[*] Photo taken from https://www.haghish.com/statistics/stata-blog/stata-programming/ascii_characters.php

Wide-area network

In 1965, Lawrence Roberts & Thomas Marill created the first Wide-area network connection via long distant dial-up between a TX-2 computer in Massachusetts and a Q-32 computer in California. A wide-area network is a large network of connected local area networks (LAN), as shown in fig 5 and is spread over the globe. During the communication, they also confirmed that packet switching offers the most promising communication model between computers.

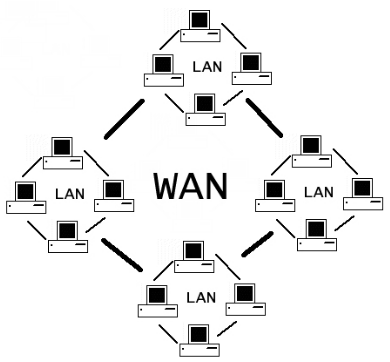


Figure 5. Wide Area Network

Initial Ideas of the Internet

In 1968, Robert W. Taylor put out his ideas about the future of the Internet and what shape it could take. J.C.R. Licklider and Robert Taylor proposed for the first time a networking experiment in which the users from one location accessed the computer at another site [14]. They explained in their research how this can be done and what could be the possible consequences while accessing a computer from another location.

The Internet prototype development started in 1969, funded by the Department of Defense, and was named ARPANET. It used packet switching to allow multiple computers to communicate over a single network. The first communication on ARPANET was done between two nodes (research labs at UCLA and Stanford) on Oct 29, 1969. The first message they sent was ”LOGIN,” but it was enough to crash the network. The Stanford research lab's computer only received the first two characters of the message. ARPANET became the center of the Internet's future and has been used by many researchers to develop new ideas around it. Initially, ARPANET used the Network Control Program for communications protocols rather than TCP/IP (explained later). In 1983, TCP/IP was incorporated in ARPANET, starting the age of the modern Internet.

In 1973, Louis Pouzin, a French Computer Scientist, designed the first packet communications network, CYCLADES [15]. This protocol was developed to explore alternatives to the ARPANET's early communication protocols and improve network research in the community. CYCLADES was the first implementation of a network protocol stack where end nodes or hosts are responsible for reliable communication instead of the network itself. This network was the first actual implementation of the pure packet model, initially imagined and described by Donald Davies. CYCLADES made significant progress in network communication, but the reduction in funding for the project due to political reasons marked its end. The European postal and telecommunications authorities chose to adopt the X.25 standard rather than packet switching as their data transmission protocol. However, in the later 1970s, the TCP/IP protocol has been developed based on the key ideas derived from CYCLADES. In the next section, we present the details of TCP/IP.

TCP/IP

The TCP/IP is the set of protocols with which two different network entities communicate. Without the TCP/IP, the data communication on the current Internet or Inter-Networking of the devices would not be possible.

The development of TCP/IP started in the 1970s by a group headed by Vint Cerf (Stanford) and Robert Kahn (DARPA) [16]. The purpose of the protocol is to allow inter-network communication between diverse networks.

Development

The experiments started with a two-network TCP/IP communication between Stanford and the University College of London in 1975. Then, In Nov 1977, a three-network TCP/IP test was conducted between sites in the US, the UK, and Norway. A computer called a router is provided with an interface to each network. It forwards network packets back and forth between them. Originally a router was called gateway, but the term was changed to avoid confusion with other types of gateways.

The early versions of TCP/IP managed both datagram transmission and routing in one protocol [17]. As the protocol grew, in 1978, the Transmission Control Program Version 3 was split into two distinct protocols, the Internet Protocol as a connection-less layer and the Transmission Control Protocol as a reliable connection-oriented service. Version 4 of the TCP/IP was abstracted into four layers of distinct protocols, as shown in 6, and is known as Internet Protocol version 4 (IPv4). This protocol was then incorporated into ARPANET on Jan 1, 1983, and is still in use on the Internet, alongside its current successor, Internet Protocol version 6 (IPv6). After IPv4, the communication between multiple networks started, and then it evolved into the modern Internet.

Key Architecture

The main principles that TCP/IP was built upon are:

The end-to-end principle Initially, the Internet started with the assumption of concentration only on speed and simplicity by putting the maintenance of state and overall intelligence at the edge nodes. Real-world needs for firewalls, network address translators, web content caches, and the like have forced changes in this principle.

The robustness principle The implementation must be conservative in its sending behavior (by sending well-formed datagrams) and liberal in receiving behavior, i.e., accept any datagram that it can interpret.

Encapsulation It divides the protocol suite into different layers depending on their functionality. The layers individually process the data and pass it to the next level. Each layer can be further encapsulated.

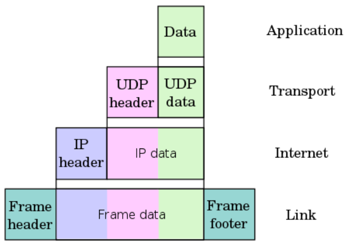


Figure 6. TCP/IP layers*
* Photo taken from https://en.wikipedia.org

The TCP/IP is abstracted into four layers, each with a particular job. The layers in the bottom-up order are:

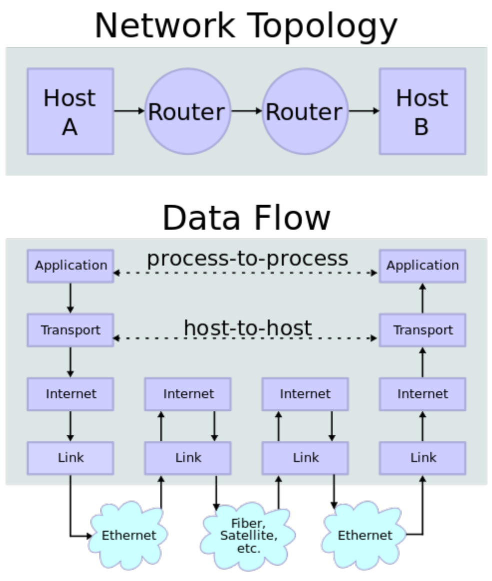


Figure 7. TCP/IP data flow [*]

[*] Photo taken from https://en.wikipedia.org

The link layer defines the networking methods within the scope of the local network link on which hosts communicate without intervening routers. This layer includes the protocols used to describe the local network topology and the interfaces needed to affect the transmission of Internet layer datagrams to next-neighbor hosts [18].

The link layer is the lowest layer of the TCP/IP model that provides the functional and procedural means to transfer data between two network entities over the physical layer and possibly correct errors in the physical transfer of data. This layer is responsible for delivering frames within a LAN. Inter-network routing is higher-level functions allowing the link layer to focus on local delivery, addressing, and media arbitration.

The data link layer is concerned with the local delivery of frames between nodes on the network's same level. Data-link frames, the protocol's data unit, do not cross the boundaries of a local area network. Inter-network routing and global addressing are higher-layer functions, allowing data-link protocols to focus on local delivery, addressing, and media arbitration by handling frame collisions.

Examples of data link protocols are Ethernet for local area networks (multi-node), the Point-to-Point Protocol (PPP), HDLC, and ADCCP for point-to-point (dual-node) connections.

The internet layer This layer provides inter-network communication, i.e., connects multiple networks through gateways. The protocols in this layer are used to transfer network packets from the sender's node, across network boundaries, to the destination node using their IP address. Internet-layer protocols use IP-based packets. This layer aims to transfer the packet to the destined network through a set of routers in between and then passes the packet to the Transport layer of the destined node (shown in fig 7), which handles further processing. The communication within a network (LAN) is handled by the lower layer, i.e., the link layer.

Primary examples of the Internet layer protocols are IPv4 and IPv6, which defines IP addresses. Its function in routing is to transport datagrams to the next host, functioning as an IP router with the connectivity to a network closer to the final data destination. The Internet Control Message Protocol (ICMP) is used for error and diagnostic functions.

The transport layer This layer provides host-to-host communication channel for the applications [19]. This layer's services are connection-oriented communication, reliability, flow control, and congestion avoidance. There are two popular transport layer protocols UDP (User datagram protocol) and TCP (Transmission Control Protocol). The UDP provides unreliable connectionless communication, whereas TCP provides flow-control, connection establishment, and reliable data transmission.

The application layer is the scope within which applications, or processes, create user data, and communicate this data to other applications on another or the same host. The applications use the underlying lower layers' services, especially the transport layer, which provides reliable or unreliable pipes to other processes. The communications partners are characterized by the application architecture, such as the client-server model and peer-to-peer networking. This is the layer in which all application protocols, such as SMTP, FTP, SSH, and HTTP. Processes are addressed via ports that essentially represent services.

The application layer is the highest abstraction layer of the TCP/IP model, and it provides interfaces and protocols needed by the users. Some of the services this layer provides are the user interface of network services, used to develop network-based applications, error handling, and recovery of messages.

The examples of the application layer protocols are HTTP, FTP, SMTP, DNS, TELNET, and SNMP (Simple Network Management Protocol)

Era of the Modern Internet Begins

Network bridging and routing

A network bridge is a computer networking device that creates a single aggregate network from multiple communication networks or network segments. This function is called network bridging. Bridging is distinct from routing. Routing allows various networks to communicate independently and yet remain separate, whereas bridging connects two different networks as if they were a single network.

In the early days, network bridging for large networks was difficult because of loops. The loops could occur when there are multiple paths in the network from the sender to the destination. The multiple paths allow the intermediate node to send the packet in any direction. The next node might return the packet expecting the packet to forward from another path. The packet goes in a loop forever, doesn't reach the destination, and creates unnecessary congestion in the network. In 1984, Radia Perlman invented the spanning-tree protocol, a fundamental concept for today's network bridges. She used unique MAC addresses of bridges in the network protocol to allow communication within LAN. The algorithm is run on all the bridges; the bridges together decide on one root bridge in the network. Every bridge maps the network and finds the shortest path to the root bridge, which prevents communication through other redundant paths.

Radia Perlman also made immense contributions to many other areas of network design and standardization, such as link-state routing protocols. She designed IS-IS (Intermediate System to Intermediate System) protocol for routing IP, which continues to flourish today. IS-IS is a link-state routing protocol. Routers share the topology with their nearest neighbors by flooding it through AS(Autonomous System). This way, every router in the network has a complete picture of the topology of the AS. Using the topology information, every router can calculate the path to any other end node through the use of a variant of the Dijkstra algorithm. While forwarding the packet to the destination through the link-state protocol, the next hop is selected based on the best path calculated to the destination. The complete topology access at each router allows the router to choose a path based on any particular criteria. It can be useful when the different quality of services are provided depending on the source or destination. However, the link-state routing protocol has scalability issues. The increase in the number of routers increases the topology updates' size and frequency and the time to calculate the path to the destination. Because of this scalability problem, it is only used to route traffic within a single Autonomous System.

Internet congestion Problem

In the late 1980s, due to many new nodes connecting to the Internet, there was a major traffic surge, and the Internet was on the verge of collapsing. Between 1988 and 1989, Van Jacobson redesigned TCP/IP's congestion control algorithms [20] to handle the congestion better and is said to have saved the Internet. The congestion algorithms are still used in over 90% of Internet hosts today.

Application Level protocols

Domain Name System

The IP address uniquely identifies every node in the network, but it was difficult for the people to remember them for every node; also, the IP addresses were not static. In 1983, Paul Mockapetris, Jon Postel, and Craig Partridge created the Domain Name System (DNS) [21], which uses domain names to manage the increasing number of nodes on the Internet by mapping the domain names of a node to their IP address. DNS is a distributed database implemented in a hierarch of name servers. Nameservers act as a directory for the nodes registered with them. In 1985, the first domain was registered: symbolics.com, a domain belonging to a computer manufacturer.

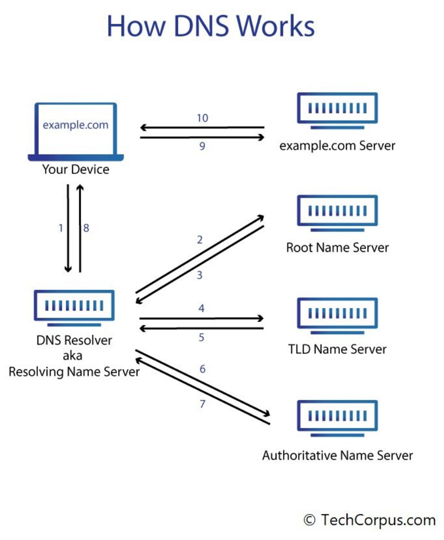


Figure 8. DNS lookup diagram *
* Photo taken from https://techcorpus.com/dns

Fig 8 explains the working of the DNS lookup process. When a client types in *example.com* in their browser, their device sends a DNS lookup request to the DNS resolver that they are registered with. In most cases, the DNS resolver of your Internet service provider (ISP). The DNS resolver likely wouldn't know the IP address of the *example.com*, unless cached from previous requests. Whenever the DNS resolver gets an unknown domain name, they query the root Name server. The root name server forwards the DNS resolver to the corresponding top-level domain(TLD) name server i.e *com* in our case by returning the IP address of the *com* name server. Then, DNS resolver queries the TLD name server, which gives the IP address of the authoritative name server (which knows the IP address of example.com in our case). In the last step, the DNS resolver retrieves the IP address of *example.com* from the authoritative server and returns to the client. Then, the client would be able to connect to the example.com

In 1998, Tan Tin Wee founded the multilingual Internet DNS and was instrumental in its internationalization. In the 1990s, under his leadership, Singapore hosted the first Chinese and Tamil websites.

In 1993, the IETF (Internet Engineering Task Force) started discussing ideas to make the DNS more secure. In 2005, they finally decided on Domain Name System Security Extensions (DNSSEC). It is a set of extensions for DNS to provide the DNS clients authentication of DNS data, authenticated denial of existence, and data integrity, but not availability or confidentiality.

E-mail

In 1986, Craig Partridge designed how e-mail is routed using domain names and named it the Simple Message Transfer Protocol(SMTP). SMTP is the basic standard for e-mail, and it still exists today since the 1980's in its original form.

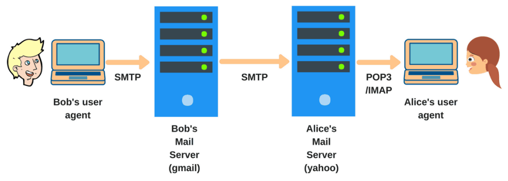


Figure 9. SMTP message flow *
* Photo taken from https://www.afternerd.com/blog/smtp

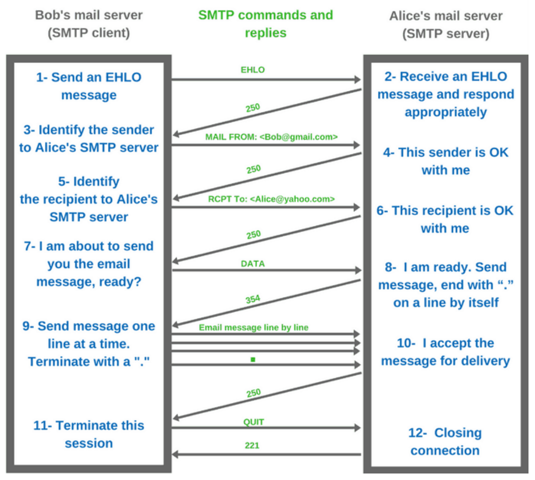

Bob's mail server (SMTP client)
SMTP commands and replies
Alice's mail server (SMTP server)
1- Send an EHLO message
EHLO
2- Receive an EHLO message and respond appropriately
250
3- Identify the sender to Alice's SMTP server
MAIL FROM: <Bob@gmail.com>
4- This sender is OK with me
250
5- Identify the recipient to Alice's SMTP server
RCPT To: <Alice@yahoo.com>
6- This recipient is OK with me
250
7- I am about to send you the email message, ready?
DATA
8- I am ready. Send message, end with "." on a line by itself
354
9- Send message one line at a time. Terminate with a "."
Email message line by line
10- I accept the message for delivery
250
11- Terminate this session
QUIT
12- Closing connection
221

Figure 10. SMTP sequence *
* Photo taken from https://www.afternerd.com/blog/smtp

When a user sends an e-mail, it goes through their mail server to the recipient's mail server and then eventually to the recipient's client, as shown in fig 9. The communication of sender's client to their mail server and between mail servers is through SMTP, as shown in fig 10. The receiving client uses the IMAP or POP3 protocol to retrieve e-mails from their mail server whenever they need it. IMAP and POP3 are the two most commonly used Internet mail protocols for retrieving e-mails.

Fig 10 shows the series of messages that are communicated in SMTP. The sender's client sends an EHLO message to initiate the communication and waits for the response before sending the next message. The sender, in order, sends FROM address, TO address, message, and termination message.

The e-mail was developed without being concerned about security. As the Internet started growing, the era of digital crime started, forcing the researchers to rethink the existing protocols and add a security layer. In 1991 Philip Zimmermann designed Pretty Good Privacy (PGP), an e-mail encryption software package that's published for free. Originally designed as a human rights tool, PGP becomes one of the most widely used e-mail encryption software globally.

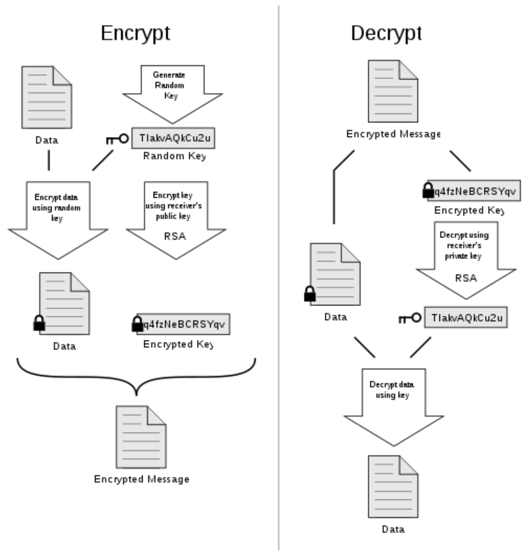


Figure 11. PGP working [*]

[*] Photo taken from https://en.wikipedia.org/wiki/Pretty_Good_Privacy

The PGP works on the concept of public-private cryptography. Every user in the system creates a public-private key pair and uploads their public key to an Internet database. When user A sends an e-mail to user B, they first generate a symmetric key Sk, encrypts the message with that Sk, encrypts the Sk with B's public key as shown in figure 11. User A sends the encrypted message and the encrypted Sk to B over the untrusted Internet. The user B decrypts the Sk with their corresponding private key and then decrypts the message with Sk. The security guarantee of public-private cryptography is that the data encrypted with a public key can only be decrypted with the corresponding private key, making PGP secure.

Search Engines

In 1989 Brewster Kahle invented the First Internet Publishing system named WAIS (Wide Area Information Server). It was one of the first programs to make searchable data through a large network by indexing it. Today's search engines are built upon the concepts of WAIS. Alan Emtage developed the world's first Internet search engine, called Archie, pioneering many techniques used by search engines today. Archie was just an index of File Transfer Protocol (FTP) sites. FTP is essentially a way to transfer files between computers. After a user finds the file through their search keyword, they need to download the file before seeing the file's content. There was no concept of natural language keywords in Archie; therefore, the users were limited to only use one word for their search.

World Wide Web

In 1990, Tim Berners-Lee and his colleagues at CERN developed hypertext markup language (HTML) and the uniform resource locator (URL), giving birth to the World Wide Web. HTML is a computer language that is invented to allow website creation. These websites can then be viewed by anyone else connected to the Internet.

In 1991, CERN introduced the World Wide Web to the public. The World Wide Web (WWW) is where URLs identify the web resources (like google.com), which may be interlinked by hypertext and accessible over the Internet. The resources of the Web are transferred via the Hypertext Transfer Protocol (HTTP). The HTTP protocol, an application layer protocol that is sent

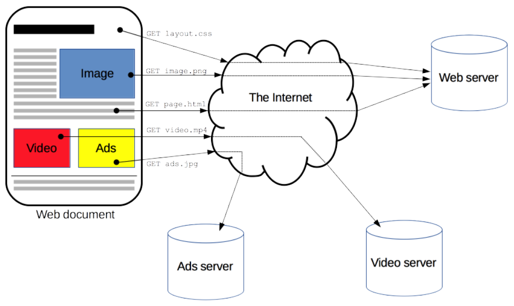


Figure 12. HTTP [*]
[*] Photo taken from https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

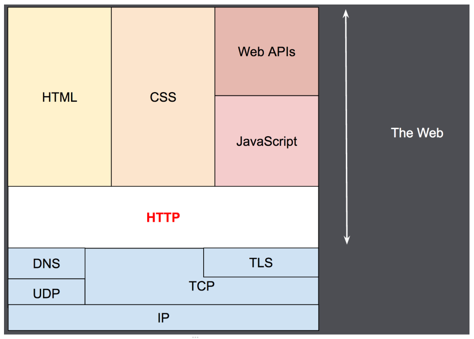


Figure 13. HTTP and layers [*]
[*] Photo taken from https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

over TCP as shown in fig 13, allows retrieving resources such as HTML documents from the Web. HTTP is the communication foundation of any data exchange on the Web. It is a client-Server protocol, where the requests are initiated by the recipient, usually through a web browser. The document is received from a variety of different sub-documents and is reconstructed by the browser using the HTTP, as shown in fig 12. A few examples of different sub-documents in an HTML page are images, videos, descriptions, texts, and scripts.

Digital Subscriber line

Digital Subscriber line is why the Internet became affordable in the 1990s and reached the entire world. In 1993, John Cioffi, known as ”Father of DSL”, developed DMT (Discrete Multitone)-based DSL technology. His idea was to fit hundreds of data channels alongside phone conversations without causing disturbances. Cioffi named the technology DMT or Discrete Multitone. However, the high interference of the lines became a problem. He built a device to switch data between channels to prevent this, creating the first broadband modem. This broadband modem worked for low rate data sending, but there was still interference while high rate data sending. Joseph Lechleider found that allocating high bandwidth in one direction and lesser in the other direction could solve interference. This idea becomes the basis of Asymmetric Digital Subscriber Line technology(ADSL) and is excellent for people who need to upload less data than download.

Later, John Cioffi also developed Very-high-speed Digital Subscriber Line (VDSL) modems, which could work with copper wires and optic fiber, making VSDL much faster than ADSL at download and upload speeds. However, the VDSL only works for a shorter distance. The ADSL and VDSL account for about 98% of the world’s more than 500 million DSL connections.

Internet Security

There has always been a concern about the security of networked computers. During the 70s and 80s, researchers with access to the ”internet” enjoyed playing practical jokes on each other through the network. These jokes were harmless but exposed flaws in the security of the ARPANET. Before the 90s, networks were relatively uncommon, and the general public was not made-up of heavy internet users, which limited the risk and threat. During these times, security was not as critical - however, with more and more sensitive information being placed on networks, it would grow in importance.

In the early 1990s, due to malicious behaviors on the Internet, researchers started improving the previous protocols with security in mind. Dr. Stephen Kent designed and developed network layer encryption, access control systems, standardized secure transport layer protocols, secure e-mail technology, Public Key Infrastructure standards, and certification authority systems.

Voice over IP

In 1973, Bob McAuley, Ed Hofstetter, and Charlie Radar developed the first voice packet over ARPANET at MIT’s Lincoln Lab. Their voice transmission used the Linear Predictive Coding (LPC), a speech analysis technique that relies on the linear predictive model to process and resynthesize compressed digital forms of voice signals and speech. In 1974, Lincoln Lab and Culler Harrison successfully transmitted test voice data packets to one another. By 1976, Culler Harrison and Lincoln Labs had a conference call over LPC. In 1982, they achieved a major milestone by using LPC to connect over a local cable network, a mobile packet radio net, and an interface with the PSTN (Public Switched Telephone Network.)

In 1988, G.722 wideband audio codec was developed, which have a much wider speech bandwidth and was also able to sample audio data two times as fast as was previously possible. The fact that G.722 offers data rates of up to 64 kbit/s makes it ideal for VoIP communication — especially those on local area networks. It was rated as ”toll quality,” meaning its audio was comparable to PSTN phone call quality.

In 1989, Brian C. Wiles created RASCAL, the first system to send voice over Ethernet networks successfully — their first VoIP application, technically speaking. Later Wiles wrote a decimation/expansion scheme that reduced the necessary bandwidth from 64Kb/s to just 32 Kb/s. He releases the program to the public domain under the name NetFone, later known as Speak Freely, which is the first software-based VoIP phone.

ACKNOWLEDGMENT

We thank Jeff Hutchings for reviewing our paper and providing valuable feedback.

REFERENCES

1. “Arpanet.” https://www.darpa.mil/about-us/timeline/arpanet.
2. “Pigeons as messenger.” https://blog.sciencemuseum.org.uk/exciting-tales-and-top-secret-work-of-pigeons-in-the-first-world-war.
3. “Pigeons: friend or foe?.” https://www.mprnews.org/story/2007/04/16/pigeons.
4. “How did people communicate before the internet?.” https://blog.eero.com/how-did-people-communicate-before-the-internet.
5. “Morse code the telegraph.” https://www.history.com/topics/inventions/telegraph.
6. “Alexander graham bell patents the telephone.” https://www.history.com/this-day-in-history/alexander-graham-bell-patents-the-telephone.
7. “Automatic telephone-dialing system.” https://www.invent.org/inductees/almon-brown-strowger.
8. “A mathematical theory of communication.” http://people.math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf.
9. “Information flow in large nets.” https://www.lk.cs.ucla.edu/data/files/Kleinrock/Information%20Flow%20in%20Large%20Communication%20Nets.pdf.
10. “Information flow in large communication nets.” https://my.ece.utah.edu/~ece6962-003/additional/Kleinrock61.pdf.
11. P. Baran, “On distributed communications networks,” *IEEE transactions on Communications Systems*, vol. 12, no. 1, pp. 1–9, 1964.
12. M. Campbell-Kelly, “Data communications at the national physical laboratory (1965-1975),” *Annals of the History of Computing*, vol. 9, no. 3/4, pp. 221–247, 1987.
13. K. Dubost, “Utf-8 growth on the web,” 2016.
14. J. C. Licklider and R. W. Taylor, “The computer as a communication device,” *Science and technology*, vol. 76, no. 2, pp. 1–3, 1968.
15. L. Pouzin, “Presentation and major design aspects of the cyclades computer network,” in *Proceedings of the third ACM symposium on Data communications and Data networks: Analysis and design*, pp. 80–87, 1973.
16. V. G. Cerf and R. E. Icahn, “A protocol for packet network intercommunication,” *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 2, pp. 71–82, 2005.
17. “Tcp/ip.” https://en.wikipedia.org/wiki/Internet_protocol_suite.
18. “The link layer.” https://en.wikipedia.org/wiki/Data_link_layer.
19. “The transport layer.” https://www.techopedia.com/definition/9760/transport-layer.
20. V. Jacobson, “Congestion avoidance and control,” *ACM SIGCOMM computer communication review*, vol. 18, no. 4, pp. 314–329, 1988.
21. “Dns.” https://tools.ietf.org/html/rfc882.

Tarun Kumar Yadav is currently pursuing his Ph.D. in Computer Science at Brigham Young University. He received his B.S in Computer Science Engineering in 2018. His current research is in network security, with a primary focus on authentication protocols.

Early History of the Internet

Barbara Chamberlin
Brigham Young University

***Abstract*—The internet was the result of several visionaries. J.C.R. Licklider was the one who originally wrote about an "Intergalactic Computer Network" and specified what that would entail. Leonard Kleinrock and Lawrence Roberts did early work on packet switching. Robert Kahn and Vincent Cerf designed the TCP/IP stack. Douglas Engelbart envisioned how the communication and collaboration would become more important than computation. Tim Berners-Lee invented the World Wide Web. All of them, as well as others, worked hard to make it a general purpose network instead of specialized and proprietary.**

■ **OUR CURRENT TIME PERIOD** is called the Information Age, defined as "The modern age regarded as a time in which information has become a commodity that is quickly and widely disseminated and easily available especially through the use of computer technology." [1] To some people, the use of computers is difficult and taxing, while to others it is so straightforward and obvious that it is practically invisible. When people discuss the cloud, and the internet, and the web, they often do not understand where one piece ends and another begins. Nevertheless, it is the internet that has made this the Information Age. If all we had were stand-alone computers, no matter how good they were, they would still be limited in the amount of data they could store, and the number of things they could do. But with the ability to interface seamlessly with other computers, we have access to more knowledge than we know how to use.

The internet was created due to the need for sharing resources. When computers were big, expensive, and rare, it was not cost effective to build them everywhere they were wanted. Time sharing was one way to allow multiple users to use the same mainframe. But if one person wanted to use multiple machines, he had to have a dedicated connection to each one. Resource sharing is still the biggest use of the internet, but it is a lot less obvious. From the beginning, it was joint efforts by government, industry, and academia working together that enabled the rapid progress of networked communication [2]. Many networks were created early on for specific purposes and groups. Some of the designers recognized the need to make a general purpose network. What we have now is the result of visionaries who actively sought to make the world a better place.

Figure 1. J. C. R. Licklider

Vision

It was during the height of the cold war when the Soviets surprised everyone by launching the

Sputnik satellite. Suddenly defense from space-based attacks became a primary concern. The Advanced Research Projects Agency (ARPA) was established in 1958 by President Eisenhower to encourage research that the military could use. ARPA had a good deal of latitude, and not much bureaucracy to deal with [3]. One hurdle that ARPA set out to solve was the communication bottleneck. The telephone system and short-wave radio were the only communication tools available. The telephone system was highly centralized and susceptible to attack. Even if the telephone system was not the target, it was still likely to be knocked out as collateral damage [3].

In October 1962, J. C. R. Licklider was named as the head of ARPA. He was a man of many talents with degrees in Physics, Mathematics, Psychology, and Psycho-acoustics. His interest in IT began when he became an associate professor at MIT in 1950, where he specialized in human factors [4]. His unique background allowed him to look at the future of IT instead of getting bogged down in the nuts and bolts of current research. In April 1963, he wrote a memo to colleagues in which he laid out the requirement for what he called an "Intergalactic Computer Network." [5] This network would allow for information search and retrieval, linked programs and information, copying of programs or data from one computer to another, storage of programs or data, and remote execution of processes. He also wrote of the need for standardized command and control language and the eventual need to search using natural language. Although he did not live to see it happen, the internet as we use it today has all of these capabilities that he described. He would have been delighted to see cloud storage and cloud computing come to fruition.

Licklider only stayed at ARPA for a few years, but his vision remained. It was obvious that the military needed this technology, not just for communication, but also for efficiency. For example, Robert Taylor, a successor of Licklider at ARPA, had several terminals in his office, each with a time-shared connection to a different mainframe [3]. Each mainframe had a different operating system with its own commands for accessing needed information. With so many computers and resources in different locations and departments, but with overlapping needs, it was clear that developing the same tools and data at each location was not cost effective. It was much better to find a way to share tools, data and resources, so he and his successors organized and funded the research that was required to make it happen.

Figure 2. Leonard Kleinrock [6]

Foundations

Leonard Kleinrock was a graduate student at MIT studying under Claude Shannon when he began studying packet switching [7]. He did not actually have a network to test his theories on, however he was able to simulate it on the TX-2, an early model transistor computer with 64K of RAM that took up most of one wall of the room [8]. He believed that messages could be sent in small pieces instead of large chunks. Optimization of message sending implied that smaller messages should be sent before longer messages. Unfortunately, there was no way to know the size of the messages. He realized that by chopping them up, they would all become short messages. If the whole message was small enough to be sent in one piece, then it would be. If it was larger than the maximum size limit, then the first fragment would get sent, and the remainder would go back to the end of the queue.

Kleinrock proved his theory first mathematically and then through simulation. He published the first paper on "Information Flow in Large

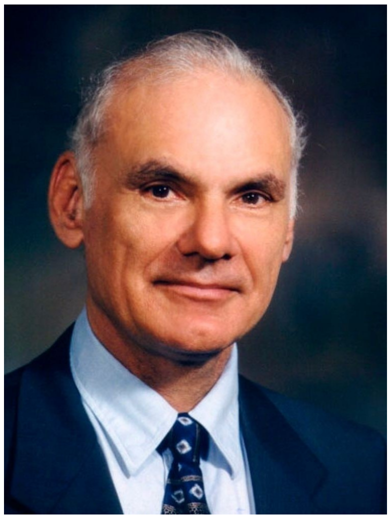


Figure 3. Lawrence Roberts [9]

Figure 4. Robert Kahn [11]

Communication Nets" in July 1961. Other researchers were studying the same thing simultaneously without anyone realizing it, and one of them, the team from the National Physical Laboratory in London, coined the term "packet switching [2]." This process is still in use today for internet communication.

Kleinrock was also friends and roommates with Lawrence Roberts. Roberts used Kleinrock's work when he created the first Wide Area Network with a colleague in California in 1965 using telephone lines [2]. They realized that it could be viable, but not with telephones that used circuit switching. In 1966 the Department of Defense (DoD) wanted Roberts to oversee their network research program. He refused to go because he recognized that it would be a management position, not engineering. The DoD told Lincoln Labs, where Roberts worked, that they would not get anymore funding unless he went to ARPA. Thus coerced, Roberts moved to ARPA specifically for the purpose of creating their networking program [10]. He published his "Plan for the ARPANET" in 1967. Apparently it was more interesting than he had expected it to be since he continued in that position for several years. His biggest problem was getting the various labs that were funded by ARPA to be willing to spend time and money on this new idea. Kleinrock at UCLA and Douglas Engelbart at Stanford were the ones who stepped up and made it happen [9].

As initially suggested by Licklider, the project was too big for one team, so the next big step was contracted out to a team at Bolt, Beranek and Newman (BBN). That team was responsible for developing the packet switching hardware called "Interface Message Processors (IMP)." Since each computer at the time had its own operating system, getting them to talk to each other was a challenge. The IMP was an intermediate step; they did not have to write a connection program to every computer in the network, they only had to write the connection to the IMP, and all the IMPs could talk to the other IMPs without trouble. The IMP was a "small" computer, only 3 feet by 7 feet, that would be provided by ARPA for each mainframe [9]. A key player at BBN was Robert Kahn. Between them, Roberts, Kahn, Kleinrock and their teams designed and built the pieces of the ARPANET, including the decision to use a distributed network architecture instead of a centrally controlled system[2]. The distributed architecture of the internet is one of the reasons that it is resilient to failure, and able to incorporate technologies that are very old and very new at the same time.

Kleinrock was the director of the Network Measurement Center at UCLA. In September 1969 his lab was the first one to acquire an IMP, and thereby became the first node of ARPANET. Soon thereafter in October, Stanford Research Institute (SRI) became the second node. A test message was attempted. UCLA tried to send

Figure 5. Leonard Kleinrock and the IMP [12]

"log" and SRI was supposed to send "in" but the receiver crashed after the second letter. Eventually, of course, they repaired it, and successfully connected both labs. By the end of 1969 UC Santa Barbara and University of Utah became the third and forth nodes respectively. Other universities and research groups joined over time, and ARPANET soon contained dozens of nodes [2] despite the fact that they did not yet have good software for controlling it.

In December of 1970, the Network Working Group finished writing the first Host-to-Host network protocol for ARPANET and called it the Network Control Protocol (NCP). It was refined over the course of the next few years, and finally made communication across the ARPANET available to applications. In October 1972, Kahn performed a demonstration of ARPANET at the International Computer Communication Conference and it was a huge success. The NCP however, had one big flaw: it was too tightly coupled to ARPANET; it could not talk to any other network [2].

Making it Useful

One of the first applications to use the NCP was email. Time-shared computers already had commands for sending private messages to other users on the same machine, so it was no surprise that people wanted to send messages to users on other computers in the network. In March 1972, Ray Tomlinson created two apps to send and

Figure 6. Vinton (Vint) Cerf [13]

read messages. It was quite dissimilar from the email systems we use today, nevertheless it did establish the '@' sign as the separator between user name and host name. Roberts expanded on Tomlinson's work to create a utility that could manage email communication. Since then email has proliferated and is one of the most common uses of the internet.

In 1972, Kahn left BBN and went to work for ARPA (which was renamed DARPA). He was designing a network that could be operated over radio and satellite, and so had a number of technical problems to solve including lost or corrupted packets, diverse network characteristics, global addressing, and others. He concluded that ARPANET's NCP communication protocol was too limited for all those reasons and more. So he started to design a new one, but quickly realized he needed help from an expert in operating systems.

So in 1973, Kahn showed up in the SRI lab at Stanford to talk to Vint Cerf. Cerf had worked with the SRI team since the beginning of the ARPANET project. Kahn and Cerf together worked to build a new Transmission Control Protocol (TCP), which they unveiled at a meeting of the International Network Working Group in September 1973. Their protocol was eventually split into two cooperating layers and renamed TCP/IP. The Internet Protocol (IP) layer is used for addressing while the TCP layer is used for packaging and provides the connection to the application. A more technical discussion will be given at the end.

TCP/IP was specifically designed to be independent of the hardware on which it was executed, thereby abstracting the implementation details from the applications. It was also intended to be non-proprietary to allow modification by users as needed. It was implemented by three different teams, and all of them were able to interface correctly and easily with each other. On November 22, 1977, the first message was sent across three networks to demonstrate the feasibility of TCP/IP. Kahn and Cerf did not patent TCP/IP; they were determined to make it freely available as soon as possible [14].

Growth

ARPANET was primarily dedicated to military purposes. Other groups saw their success and created their own networks dedicated for their own needs. None of these were for general purpose use and they were unable to communicate with each other. Kahn, Cerf and others started working to make internet connectivity available to the public, through small and steady (but disjointed) efforts. An important point came when David Clark and his MIT research group decided to make TCP/IP work in personal computers[2]. Then, at UC Berkeley, a team of researchers were able to incorporate TCP/IP into the Unix kernel. AT&T was actively distributing the Unix operating system, and it became the most common OS for mainframes at the time. That change made the internet available to a much larger community. Unix is also the precursor to the Linux operating system that is now the most common server on the internet.

Shortly after TCP/IP was integrated into Unix, in 1979, two graduate students at Duke University, Tom Truscott and Jim Ellis, came up with a plan to allow messaging between universities. They enlisted the help of Steve Bellovin at the University of North Carolina, and invented USENET, which became free for system administrators in 1980. This became very popular, and was the most common way to communicate across the nascent internet [15].

With the proliferation of networks and computers, it became impossible for every computer to maintain an up-to-date lookup table of all the possible addresses. The internet was re-organized into regions and in 1984, Domain Name Server (DNS) system was invented to provide lookup tables for networks that were under them in the hierarchy, with a few root servers providing connection between regions. This simplified communication by allowing people to use named addresses instead of numbered addresses, with routers using DNS to find the correct destination.

In 1985 The National Science Foundation (NSF) created a network for higher education called NSFNET that was designed to link universities together. One of their requirements for a university to join was that the network would be available to all qualified users, not just computer engineers. It was also mandatory to use TCP/IP. The NSF, DARPA and other federal agencies worked together to distribute the cost and ensure that the internet would function smoothly despite the diversity of uses [2]. They established several federal task forces to maintain it. The NSFNET actively encouraged universities to link with local businesses and interests, but they refused to allow commercial or private traffic on a national level.

Commercialization

The first commercially available connection to the internet, called Telenet, was established in 1975 by BBN. It required dedicated lines and cost a monthly fee, but, once connected, access was unlimited. It used the same architecture as ARPANET. Other commercial internet service providers (ISP) such as PSI, UUNET, ANS CO+RE and more began popping up, but they were still limited by governmental restrictions. The first commercial, dial-up, text only ISP, called The World, became available in 1989 [16].

In 1988 the NSF initiated discussions about the commercialization of the internet. As more businesses began taking an interest in the internet, they also began to participate in the task forces that maintain and define it. These discussions culminated in 1995 when NSF relinquished control to private companies [2].

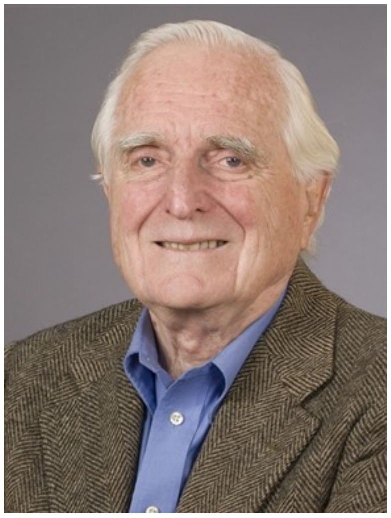


Figure 7. Douglas Engelbart [18]

The Need for Speed

Although ISPs were starting to proliferate, and people were becoming familiar with the internet, it was still a very slow process. To download a low-quality song took from 10 to 30 minutes depending upon your connection speed. To download a movie took all day at a good speed, and 5 days at a low speed [17]. In 1988, Kleinrock and other UCLA professors submitted a proposal to congress called “Toward a National Research Network.” Senator Al Gore, who had been pushing funding for internet research since the early 1970’s, used their proposal to write a bill called “The High Performance Computing and Communication Act,” aka “the Gore bill.” This was signed into law in 1991 and provided $600 million for the National Research and Education Network and for development of gigabit/sec computing. That research of course led to the “Information SuperHighway”.

World Wide Web

At the same time that Kleinrock and Roberts were inventing the foundations of the internet, Douglas Engelbart was inventing the foundations of the World Wide Web. His lab at SRI, called the Augmentation Research Center, was dedicated to discovering ways for computers to help people collaborate with each other. Out of all the labs funded by ARPA, SRI was the only one that was eager to join the ARPANET, which is how they became the second node [9].

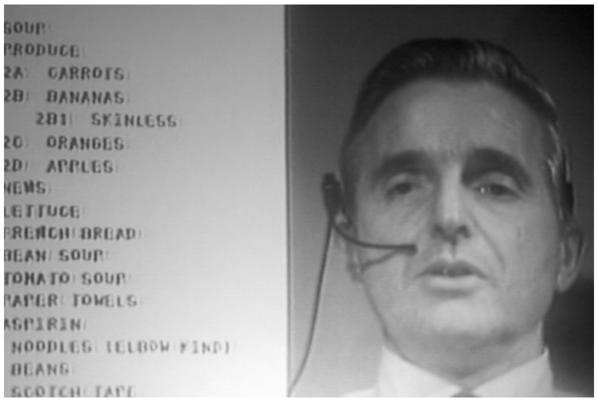


Figure 8. The Mother of all Demos [19]

Engelbart created a hypertext system called oN-Line System that was designed to encourage collaboration and knowledge sharing. Sometime between 1962 and 1968 he invented the computer mouse, the hyperlink, the Graphical User Interface (GUI), and the word processor, and founded the field of human-computer interaction. In 1968 he demonstrated all of it to a group of engineers at an event that has become known as the “Mother of all demos,”. Many of the engineers in attendance were impressed, but did not see how it applied to reality. They were still focused solely on computation, and did not know what to do with his ideas. Due to the lack of comprehension he was unable to get funding to develop them further. Many of his students went to work at Xerox and Xerox shared their innovations with Apple. But those companies were more focused on individual productivity, and less on collaboration. Apple and then Microsoft further developed some of his ideas such as the mouse, word-processor, and GUI into many of the tools that we are familiar with today [20].

Independently, Ted Nelson was also developing a hypertext system, and in 1967, he and Andreis Van Dam created the Hypertext Editing System. The next year, Van Dam wrote another hypertext system, called File Retrieval and Editing System which was based on Englebart’s NLS, and which was used as a word processor and collaborative teaching tool by the Humanities department at Brown University. But most people who were familiar with hypertext thought it a niche application, and believed it would languish in obscurity.

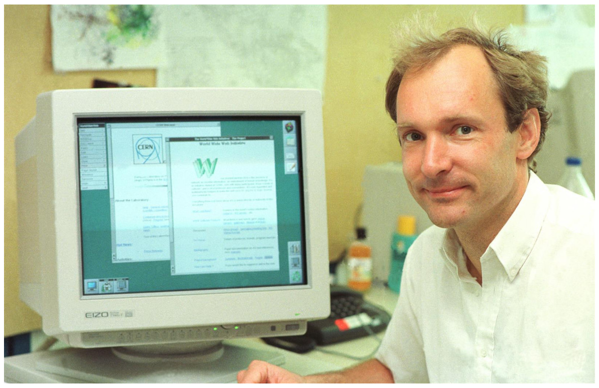


Figure 9. Tim Berners-Lee[21]

By the late 1980s, PCs were in many homes and most businesses. Larger companies had Local Area Networks and some had access to USENET or NSFNET. Tim Berners-Lee was a computer scientist at the CERN particle accelerator in Switzerland. He realized that scientists were experiencing difficulty collaborating since the information was stored in too many different formats on too many different computers. He designed a solution based upon the existing technologies of hypertext and internet, and wrote a proposal in 1989 for a system that would link all the information together. His proposal was rejected as too vague. Despite that, he was eventually granted time to pursue it. During September and October of 1990 he created the first versions of Hypertext Markup Language (HTML), Uniform Resource Locator (URL), and Hypertext Transfer Protocol (HTTP). He wrote the first Web page editor and browser in one app called WorldWideWeb.app and the first web server (HTTPd). In early 1991, he and others from inside and outside CERN were adding pages to the World Wide Web. He realized that in order for the technology to be adopted by the general public, it would have to be easily available. So he and others convinced CERN to make the code publicly available and free [22].

Internet of Things

Naturally, other fields in computer engineering were developing at the same time as the internet. Miniaturization led to a variety of wearable electronics, and the dream of connecting them to a system with more power was always there. The first patent for Radio Frequency Identification was done in 1973. The first non-computer object to be networked into a computer was a vending machine at Carnegie-Mellon. The Computer Science department wired it into their mainframe, and wrote an application to determine if drinks were available and cold. In the early 1990's, several researchers started using radio transmission to link a variety of wearable devices to their computers. In 1999 the Auto-ID center was created at MIT for researching how to allow RFID to access the internet. In 2005, the Interaction Design Institute Ivrea, in Italy designed a handheld micro controller for students to use in their projects [23]. It's still a developing field, but more and more people are using smart watches, smart homes, and linking it all into their smart phones.

Technical discussion of TCP/IP

TCP/IP defines a layered approach to communication. The lowest, network, layer includes the physical hardware as well as the drivers for that hardware. Each computer, cell phone, router, wireless transmitter, wireless receiver, or other piece of hardware has its own driver, with detailed instructions about how that machine should handle data. This level of the communication stack does not access the actual data, it just transmits it.

Data at this level is included in an Ethernet Frame. The first item in the frame is a destination MAC address which is six bytes long. Next is the source MAC address. The EtherType field is two bytes long and can indicate either the size of the payload, or the protocol to be used at the next layer. The payload, which is the actual data transmission, is the next item in the frame, and can range from 46 bytes to 1500 bytes. Finally a Frame Check Sequence (FCS) is transmitted. It is four bytes long and is used to verify that the data received is correct. The receiver calculates the expected FCS and compares it to the transmitted FCS. If they are different, then the data is presumed to be corrupted, and is re-transmitted [24].

After a data frame has been received and verified, the Ethernet Frame header is stripped off, and the payload is passed up the stack to the next layer. The network layer uses the IP or Internet Protocol. The IP section is the part that really makes the internet function. Its primary

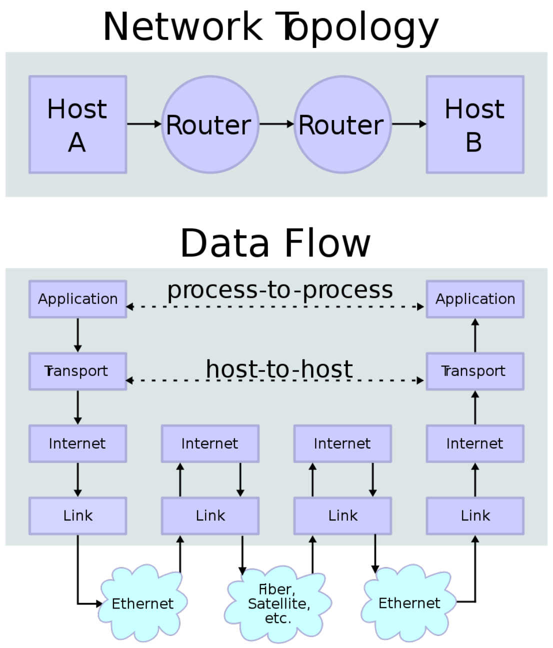


Figure 10. TCP/IP stack [25]

function is to be a selector. It is the component that determines the destination of a packet, and forwards it. If the destination is another machine, then it selects the correct output line, and sends it onward. If the destination address is the local machine, then it removes the IP header and evaluates it to determine which component of communication stack should receive the payload. A machine that only forwards to other machines, never to applications, is called a router [26]

The next component up the stack is either TCP (Transport Control Protocol) or UDP (User Datagram Protocol). TCP is the software component that controls the connection information. When TCP receives a notification from IP, it determines which application needs that data, strips off the TCP header, reassembles the packets in the correct order, and delivers the complete data to the application.

When an application has a message to send, TCP first attempts to establish a connection. It uses a synchronize and acknowledge process or 3-way handshake. First the client TCP requests a connection and transmits a random number (A). This is the SYN or synchronize step. The server accepts the connection by sending back the original random number incremented by one (A+1) and another random number (B). This is the SYN-ACK step. The client then sends both numbers incremented by one (A+1, B+1) for the final acknowledge (ACK) step. Once the connection is established, The TCP program breaks the data into packets and sends them one at a time. It then checks to make sure that all the packets arrived without error, and repeats any that failed.

The handshake, synchronization, and error handling take a significant amount of time and bandwidth, but if data integrity is necessary, it has to be done. On the other hand, if lost or corrupted data is not a big deal, such as when streaming movies and music, then the application can use UDP instead of TCP. UDP forwards data packets to the correct application without any error checking or handshaking.

Conclusion

Once, the internet was science fiction. Now it is reality. It's not just about computation anymore; it is about communication and collaboration, and it's in everything. Licklider knew it was possible. Engelbart tried to make it happen. Kleinrock, Roberts, Cerf, Berners-Lee and others made it their life's work, not just to create the technology, but also to see that it is used for the good of humanity.

■ REFERENCES

1. "Information age.," 2020. In Merriam-Webster.com dictionary. Retrieved November 2, 2020.
2. B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. Wolff, "A brief history of the internet," vol. 39, no. 5, 2009.
3. K. Mayo and P. Newcomb, "How the web was won," *Vanity Fair*, January 2009.
4. Wikipedia contributors, "J. c. r. licklider," 2020. [Online; accessed 31-Oct-2020].
5. J. C. R. Licklider, ""memorandum for members and affiliates of the intergalactic computer network"," Apr 1963.
6. "Leonard kleinrock frontiers of knowledge laureate," 2015.
7. "History-computer: Leonard kleinrock."

8. C. Metz, “Leonard kleinrock, the tx-2 and the seeds of the internet,” *Wired*, Oct. 2012.
9. M. Weber, “2017 chm fellow lawrence g. roberts,” *CHM blog*, April 2017.
10. C. Metz, “Larry roberts calls himself the founder of the internet. who are you to argue?,” *Wired*, September 2012.
11. M. Geselowitz, “Robert kahn, an oral history,” 2004.
12. L. Kleinrock, “Personal history/biography: the birth of the internet,” June 2010.
13. v. cerf, “Vint cerf ”the future of the internet of things: Desirable properties of an iot ecosystem.”
14. R. Singel, “Vint cerf: We knew what we were unleashing on the world,” *Wired*, April 2012.
15. “Usenet newsgroups history.”
16. K. A. Zimmermann and J. Emspak, “Internet history timeline: Arpanet to the world wide web,” *Live Science*, June 2017.
17. B. P. Eha, “An accelerated history of internet speed (infographic),” *An Accelerated History of Internet Speed (Infographic)*, September 2013.
18. I. Thomson, “Douglas engelbart, pc pioneer and creator of the mouse, dies at 88,” *The Register*, July 2013.
19. C. Metz, “The mother of all demos — 150 years ahead of its time,” *The Register*, December 2008.
20. V. Landau, “How douglas engelbart invented the future,” *Smithsonian*, Jan. 2018.
21. “A short history of the web.”
22. “History of the web.”
23. G. Press, “A very short history of the internet of things,” *Forbes*, June 2014.
24. Wikipedia contributors, “Ethernet frame,” 2020. [Online; accessed 9/12/2020].
25. Wikipedia contributor Kbrose, “Internet protocol suite,” 2020. [Online; accessed 10/12/2020].
26. T. Socolofsky and C. Kale, “A tcp/ip tutorial,” January 1991.

Barbara Chamberlin is a graduate student of Computer Science at Brigham Young University. She also earned her BS in Computer Science from BYU. She worked for several years as a programmer before returning to school. Her current research is focused on fault tolerance in distributed computing. She is also interested in Software Engineering, Computer Science education, and Software Design.

Representational State Transfer for a Modern Web

Matt Pope
Brigham Young University

***Abstract*—As the web grew, it had need to evolve to support the ever-growing number of users. The original architecture for the web didn't support this type of scaling and had room for improvement in many areas. During the standardization process, representational state transfer (REST) was derived from constraints found in existing web architectures. REST was used as a tool to evaluate new proposals and identify areas that needed improvement. Since then, the ideas from REST have been borrowed by engineers and applied in other domains, such as in the development of application programming interfaces (APIs). Understanding the critical parts of REST and its conception can help engineers build networked applications with similar properties and appropriate constraints.**

■ **THE TERM** RESTful APIs is used by the engineering community, though it seems like not too many people have a coherent idea of what it means. They might have picked up a part, here and there, but have a hard time forming a well-defined concept. To dispel some of these misconceptions, this article visits the past to the birth of REST to get a picture on the forces that helped create it, the principles that it focused on, and how it helped establish the modern web architecture.

This paper focuses on the network-based architectural aspect of REST rather than on the complete picture that its inventor, Roy Fielding, envisioned. Precisely, there are pieces of REST, such as the role of resources, links, views, and the concept of hypermedia as the engine of application state that will at most be spoken about at a high-level. Instead, this paper examines the set of constraints and relationships between parts of the system that are valuable for network-based systems and the advantages and disadvantages of them.

The original dissertation by Roy Fielding [1] was written in the late 1990s to early 2000 and includes terminology which may be unfamiliar or has changed since then. In fact, Fielding's commission during standardization of the early web resulted in a redefinition of a few terms that were commonly used, like resource. Thus, even terms used in academic papers or early web documentation prior to the dissertation may have different or unexpected meanings. The terms used in this paper are mostly pulled from the vocabulary that Fielding himself used, so that a reader of this article could quickly comprehend his dissertation if they wanted to read it for themselves.

The important terms to keep in mind are included here. A component is "an abstract unit of software instructions and internal state that provides a transformation of data via its interface". An architectural style is "a coordinated set of architectural constraints that restricts the roles/features of architectural elements and the allowed relationships among those elements within any architecture that conforms to that style." The architectural elements in a style include components and connectors, which are "an abstract mechanism that mediates communication, coordination, or cooperation among components." To summarize, data in a system flows through connectors to reach components. The constraints and allowed relationships on the connectors, components, and other elements describe a style.

In talking about properties of architectures, Fielding used words like scalability (the ability to support large numbers of components or interactions between components) and simplicity (maintaining a separation of concerns to where functionality is implemented), whose definitions haven't changed and which are fairly common. Other words like visibility, the ability of a component to monitor or mediate the interaction between two other components, or evolvability, the "degree to which a component implementation can be changed without negatively impacting other components" [1], may be less well-known.

This paper begins by examining the landscape of network architectures that Fielding and other engineers would have been exposed to. It discusses the properties, constraints, and advantages of each to make clear why they were chosen or left out of REST. Using this foundation, this paper shows how Fielding derived REST as a combination of existing architecture constraints and then helped push the Internet's architecture forward with additional constraints that would allow it to scale. This path can be enlightening to engineers who are interested in deriving their own, possibly novel, architecture from the constraints that are in their own domains.

A Survey of Existing Network-Based Architecture Styles

When Fielding started his investigation into how extensions should be made to HTTP, he realized it was critical to get to know the existing field of networking architectures. It is useful to look at these architectures as the group from which design lessons and principles may have been elicited. Additionally, it was these types of architectures that engineers were using in the late 90s – architecture's whose futures were sometimes touched by Fielding's work. Some of these architectures have clear origins and engineers who are credited with their development or popularization, while other architectures may have a murky origin, arising from unknown or multiple sources.

We refer to these architectures by their style – the set of constraints that restrict the roles and features of architectural elements and relation between those elements. This is the same way that Fielding analyzed the architectures when understanding the properties they promoted. The ordering of these styles is arbitrary, but they are clustered by how close they may be related.

Data-flow and Replication Styles

The pipe and filter style has each component read in streams of data, optionally transform and process the data, and then output a stream of data, typically before it has even finished ingesting data. One constraint is that each component must be completely independent of other components; they cannot share state or control with each other. This constraint allows for a simple system. The final output of such a system is a composition of each component. This style promotes a variety of traits like reusability (arbitrary components can be connected together if the data being transmitted is of the same format), extensibility (easy to add new components), verifiability (easy to analyze), and concurrency (generally quick to start outputting data). Downsides include that each component added to a pipeline increases latency, it is hard to interact with data mid-way through the pipeline, and that each component is limited in its interactions with other components and its own environment.

The uniform pipe and filter style builds on the pipe and filter style with an additional constraint that each component has the same interface. This constraint allows components to be arranged in arbitrary orders and simplifies understanding how some component works. The cost to this style comes with reduced throughput since data needs be converted to and from the uniform interface into whatever internal style is utilized by the component. Doug McIlroy is credited for having originally proposed this design while a manager at Bell Labs in 1972. It was his employees and co-creators of Unix, Ken Thomson and Dennis Ritchie, who actually implemented what would become known as Unix pipelines and limited the architecture to a linear pipeline [2].

The cache style is a variant of the replicated repository style, a style that isn't included in this not comprehensive list. The cache style increases accessibility of data and scalability of services that it is in front of. A cache can reduce the latency of requests and enable some level of protection against failures in the server. They provide the illusion that clients are interacting with a single, centralized service. Caches are typically easy to implement and are generally only beneficial to a system.

This style can be traced back to Sir Maurice Wilkes, who was a Professor at Cambridge in 1965, who wrote a paper on cache memory, though with a different, politically incorrect name [3]. The actual term cache came about a few years after thanks to an editor of the IBM Systems Journal who wanted a more concise term than ”high-speed buffer” which was used by actual computing systems.

Hierarchical Styles

The client-server style is probably the most common type of style for networked applications. A client makes requests to some server, which responds with a rejection or the content that was desired. The overarching constraint is separation of concern: the server component is simplified to the point where it can scale easier and the client typically handles the user interface functionality. Since concerns are handled separately, they can evolve and be updated separately. This style does not constrain how application state is separated between the client and the server.

The client-server style is one of the oldest styles, dating back to when early computers could finally multitask and computers began to spread out. The distance between a component (client) and the data it wanted to access (on a server) became bigger, and the number of components wanting to access that data increased. Most attribute the actual formalization of the client-server style to engineers working on ARPANET in the 1960s and 1970s [4], [5], without specific names.

The layered-system style is composed of layers that call operations on lower layers and provide operations for above layers to call. When used in conjunction with the client-server architecture, it forms the layered-client-server style. The main constraints are that layers reduce coupling by hiding layers from those that aren't directly above them. This improves reusability and evolvability at the cost of additional latency and overhead. The layered-client-server includes the ideas of proxy (a shared server for many clients) and gateway (a forwarding server) intermediary components.

From these past few styles, several more can be derived. Their names are intuitive, if a bit verbose. The client-stateless-server style derives from the client-server style. It has a single additional constraint that session state must not be in the server. That is, each request to the server must include all the necessary information to satisfy the request and not depend on any saved state on the server. This constraint increases visibility, reliability, and scalability by sacrificing network throughput.

The client-cache-stateless-server style is derived from the client-stateless-server style and the cache style by including a cache component. The advantage to this style is that the cache allows for faster response times since some interactions between components can be eliminated. The layered-client-cache-stateless-server style derives from the layered-client-server style and the client-cache-stateless-server style and contains all the constraints and components of both, such as layers and intermediaries.

Mobile Code Styles

Differing from replication, data-flow, and hierarchical styles, mobile code styles use the fact that data and processing of that data can occur in different physical locations [6]. Often, interactions between components with a very close physical location are considered to have low or zero cost in contrast to interactions between components that are communicating through a network over some distance. Examples of this style include virtual machines which execute code in a controlled environment (used commonly in scripting languages) and remote evaluation which allow clients to send an operation and the data required by that operation to a server where it is executed (for example, in rendering).

The most common mobile code style today, however, is the code-on-demand style and its derivatives, where a client can request a server for code to be executed locally. This allows the client to initially only include a base set of functionality and be extended at a later time. This style allows the server to have improved scalability since it can offload work to the client. It also allows the client's static functionality to be simpler, though at the cost of a decrease in visibility since code is now spread across the client and server. One common use case of this style is in web browsers with the delivery of JavaScript (or in Fielding's time, Java applets). The layered-code-on-demand-client-cache-stateless-server is a not just a mouthful, it is also the style derived from the code-on-demand and the layered-client-cache-stateless-server styles inheriting both its advantages and disadvantages.

Peer-to-Peer Styles

The last style group included in this overview are the peer-to-peer styles, which consist of a set of looser constraints than the client-server style. In the event-based integration style, a component broadcasts events which can be listened for by other components. This reduces coupling between components since components register through the system and the system is the one invoking the registered components. Such a design allows for extensibility and code reuse, though at the cost of poor understandability (it is hard to know what will happen when an event is broadcast) and poor failure recovery (what happens when a partial failure occurs). The peer-to-peer style was popularized by Napster in the late 1990s, but one of the first implementations was USENET in 1979, and the idea certainly dates back much farther than that.

Additional styles in this group exist like C2, which combines event-based integration with a layered-client-server style, and the distributed object style, which uses a set of components interacting together and a well-defined interface for operations that can be invoked on objects in the system. The latter was used to derive the brokered distributed objects style, which is often considered the very first version of the popular service oriented architecture (SOA). These styles in the peer-to-peer style group tend to suffer from common disadvantages, namely those of reliability, due to their distributed nature.

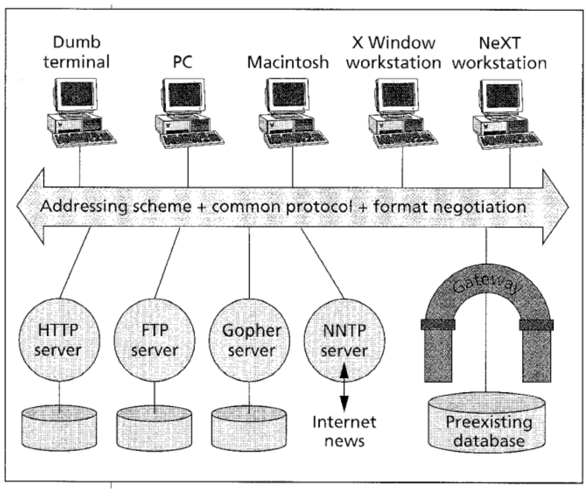


Figure 1. Diagram of the early web architecture from 1990 describing how a variety of clients read and manipulate information on a server through a set of common standards [7] .

Early Web Design Principles

The web is the shared information space of both humans and computers [7]. This overarching goal that Tim Berners-Lee envisioned created many of the familiar properties of the web that we still see today. For some time, the audience of the web was primarily university and government labs. The information on that web was generally research data, notes, and contact information spread across lots of types of computers. The web needed to grow to support more than just this audience and these types of data; it needed to have a consistent interface to any type of structured data that was able to run on all platforms. An architecture that could grow uninhibited.

The web's architecture [7] was proposed in 1989 by Tim Berners-Lee while he was at CERN – a European research organization. It (**Figure 1**) described the existence of flexible specifications to ensure interoperability, uniform resource identifiers to know where data existed, HTTP to know how to retrieve the data, and HTML to richly display and interconnect data. The Internet Engineering Task Force and World Wide Web Consortium began to coordinate the development of early requests for comments (RFCs). These documents would describe how the initial specifications and implementations could be extended. For example, it was RFC 2068 [8] that would first describe HTTP/1.1 in 1997. Later modifications would update and replace the HTTP/1.1 specifications, like RFC 2616 [9] in 1999 and RFCs 7230 to 7235 in 2014.

By the time that HTTP/1.0 was released in 1996, there had been close to 2,000 RFCs submitted. It had become clear in the early nineties that the initial network characteristics of early HTTP were insufficient to handle the rapidly growing web and would need new ideas to tackle issues that would only worsen over time. Evaluating the proposals (and ideas before they became proposals) to determine how well they aligned with the web's principles was a challenge – one that Roy Fielding tackled with his work from 1996 to 1999, eventually culminating in his 2000 PhD dissertation titled ”Architectural Styles and the Design of network-based Software Architectures” [1] at UC Irvine.

Early web architecture was based on principles such as separation of concerns, simplicity, and generality. Fielding hypothesized that the early web architecture had several implicit constraints that explained its design rationale. In order to extend the web, you simply could identify properties that are desirable, select the architectural styles that induce those properties, and then combine them to the web's existing style. This formed the basis for his second hypothesis: that the properties of a modern web architecture could be derived by joining constraints to the existing web architecture style.

His final hypothesis spoke to how he could evaluate proposals to modify the web architecture. Simply compare proposed extensions against constraints within the style. A conflict would indicate that some design principle behind the web was violated. That conflict can be handled by rejecting the proposal (to be developed outside, but parallel to the web) or amending it to better align with the accepted design principles. However, if this conflict was necessary for the modern web, a specific indicator could be used.

Born from the NULL style

Fielding's plan was to "use an architectural style to define and improve the design rational behind the Web's architecture, to use that style as the acid test for proving proposed extensions prior to their deployment". To get to that architectural style though, rationale had to be provided for each of the constraints. It was while working on the standardization process for HTTP that he distilled this style [10].

Instead of starting with some existing architectural design and adding constraints, REST was created from the nothing, the null style, and adding familiar components until it satisfied the needs of an intended system. Fielding claimed that this type of process favored creativity and unboundedness over system restraint and understanding the context of the system. The initial null style is equivalent to a system without distinguished boundaries between components.

Catching up to existing architectures

The client-server architectural style was added to the null style. The valuable constraints in this style were driven by the separation of concern principle – separate the user interface concerns from the data storage concerns, allowing for the user interface to be portable and the data storage to be internet-scale. This separation allowed for independent evolution of web browsers and servers, which is a critical part of how new web technologies have been brought to market.

Add the stateless constraint, to form the client-stateless-server style. Stateless communication requires that requests contain all the information necessary to understand the request. This style promoted the properties of visibility, reliability, and scalability, which are critical parts of the modern web. The full nature of a HTTP request can be determined by the request itself, recovery from partial failures is easier, and scalability is improved since the server doesn't have to store state between requests. Even implementation is easier because server resources don't need to be managed across requests.

Then, add cache constraints, to form the client-cache-stateless-server style. The cache constraints require that data within a response is labeled as cacheable or non-cacheable. If its cacheable, then the client can reuse that response for future equivalent requests. The cache improves scalability, efficiency, and performance. It is possible that a cache could serve stale data, a trade-off that would decrease reliability, but it is often considered necessary.

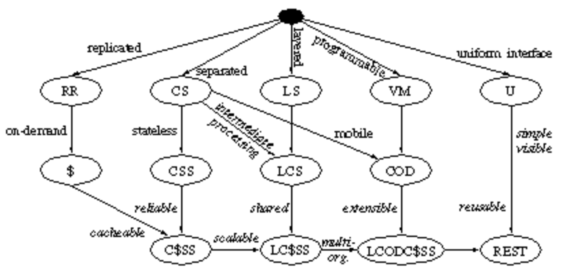


Figure 2. The REST derivation as described by Roy Fielding, is made up of styles with constraints that complement each other [1].

The Modern Web

At this point, the client-cache-stateless-server style contains the set of constraints seen in the early Web architecture, in Figure 1. The modern web needed extensions to add additional constraints for new features that were outside the early design.

The most central theme of these additional constraints is the emphasis on a uniform interface between components. Instead of each component specifying how you can interact with it, they each follow a general contract, which ensures that the overall system architecture is simplified and visibility of interactions is improved. There are four individual constraints that REST defines for interfaces, which work together to create a uniform interface. They are 1) identification of resources, 2) manipulation of resources through representations, 3) self-descriptive messages, and 4) hypermedia as the engine of application state.

As previously mentioned, there are some downsides when using a uniform interface, though REST considers these acceptable. Efficiency is lost because now applications need to translate data to and from a standardized form, rather than whatever internal format might be most useful. Small-grain hypermedia data transfer suffers as well, for similar reasons. In the early web, this wasn't a large problem since it was not considered the common case but has recently grown in time. (A problem that helped encourage the development of HTTP/2 in 2015 and HTTP/3 in 2020.)

Fielding considered that other constraints are needed to improve application behavior. For example, the layered system constraints are also included, to fix component behavior so that each component can only see the layer they are interacting with. The constraints help improve system scalability [11], reduce system complexity and promote independence, at the cost of some overhead and latency [12]. Layers are useful when designing networked application because they can encapsulate legacy services, protect against legacy clients, and allow for shared caching and load balancing with intermediaries [13], [14].

With these constraints as the basis of the modern web architecture, Fielding proposed one further optional constraint set: those from code-on-demand style. This style was considered optional because of how it can reduce visibility since client functionality can be extended by executing code. He considered it necessary though, since it allowed clients to become drastically more simple (features are no longer required to be pre-implemented) and allows the development of an architecture that supports some desired behavior as the general case, but an understanding that it may be disabled within some contexts (typically referred to as graceful degradation). Since this set of constraints was optional, it is simple to understand that architectures that chose to not include them don't suffer from their disadvantages.

It is satisfying to see that the modern web architecture, that is the architecture style that REST proposes, shown in **Figure 2**, is concretely founded upon existing architectures that historically worked well to fit the properties and goals of the web. The additional constraints that REST included are critical to scale the web up to what it has transformed into. Uniform interfaces and a layered system mean that new components scale, are visible, and work well together. While it is certain that the modern web architecture may evolve further, these constraints have helped push it in a positive cohesive direction rather than fragment it further.

Influence on Early Specifications

REST's influence can be seen in the design and development of the architecture of the modern web as early as 1994. Fielding's dissertation speaks about the experience he had and lessons he learned applying REST as he was working on standards for HTTP and Uniform Resource Identifiers (URIs) and as this technology was deployed in software packages like libwww-perl and the Apache HTTP Server Project.

Prior to REST, the web's architecture was described only in a couple of places and was severely out of date since it was evolving so quickly. These documents were limited to unorganized notes [15], a few scholarly papers that introduced the web [16], [17], some draft proposals for new features, and a mailing list that was used as a discussion group by interested parties. The task to standardize the web interface protocols was assigned to the Internet Engineering Taskforce (IETF) and supported by the World Wide Web Consortium (W3C) which was formed by Berners-Lee.

Roy Fielding was tasked to write the specification for the Relative URL, and cowrite HTTP/1.0. Eventually he became the primary author of the HTTP/1.1 specification and URL to URI revision specifications. It was during the development of these specifications that first concepts of REST were developed, though initially they were called the HTTP object model and were used as a way for communicating web concepts between engineers. It would evolve over the next few years and finally gain the name Representational State Transfer. Such a name helped describe how a web application, as a distributed hypermedia system, should behave. A network of pages which can be traversed by selecting links resulting in a new page being shown to a user: effectively a state machine.

It is not uncommon to wonder what precisely was influenced by REST after hearing so much about it. The Uniform Resource Identifier (URI) is likely the most well-known of these things. The concept that URIs now captured had been known by many names, including www addresses, universal document identifiers, and most commonly universal resource locators (URL).

That original definition of URIs were simply document identifiers, usually meaning the location of the document on some network. This proved to be inadequate as services were introduced into the web. A user may be interested in a service or the result from some invocation of the service rather than a document. Additionally, the old definition suggested that the identifier was naming the data that was transferred and that every time this data would change it might receive a new identifier. There also arose the need to provide addresses for documents that did not exist (but might in the future) and provide addresses in a way that only applied to naming some resource, rather than locating it.

REST redefined the term resource for the new URI standard, in RFC 2396, to be whatever an author intends to identify rather than whatever the author was identifying when the URI was created. This means that a reference may remain static even if the content that the reference returns changes over time. This change has impacted almost all systems that have and use dynamic content. This new definition has also clarified that the content that is transferred is also not the resource, but rather a representation of that resource.

The HTTP specification was also influenced by REST. When REST was used to identify problematic areas in HTTP, it found several areas for improvement, such as planning for the deployment of new protocol versions, separating message parsing from HTTP semantics and the transport layer, improving cache control, and ensuring that all parts of the protocol are self-descriptive. From these problem areas, arose elegant solutions.

The HTTP protocol now can permit clients and servers to do some (basic) protocol negotiation and decide which features to use and which features the client supports. Extensible protocol elements now permit more specific or even brand-new error codes, which, if the semantics were unknown by a client, could be interpreted by clients as an error of the same class (i.e. a status code 435 could be treated as status code 400). HTTP/1.0 and HTTP/1.1 included the target URL's host information in a header field, which allows for multiple domains to live on the same server. HTTP/1.1 includes several response headers for cache control, content age, and an etag.

Not only did REST identify improvements and help justify why they were necessary, but it also helped reject critical proposals. One such proposal was a new set of methods (MGET, MHEAD, etc.) for batching multiple requests within a single message. It violated several constraints: it required that clients know all of the requests to be included before it could write to the network due to the content-length header field, it required that intermediaries (such as caches or proxies) be able to read all of these requests to determine which they could resolve locally, and complicated how access-control would work when a subset of these requests would need to be denied. Another proposal that was rejected was write-back caching: caching of PUT requests to satisfy a later GET response.

Despite the impact of REST on the web architecture and specifically HTTP, there are some mismatches that existed after Fielding's work on HTTP/1.0 and HTTP/1.1. The largest of these is the use of cookies, which allowed data to be passed without sufficient semantics and allow a user to be tracked as they browse between sites. There is also some trouble when differentiating between authoritative responses, from a real data source, and non-authoritative responses, like those from a cache. It is perhaps best to view REST as a well constructed guideline, rather than an absolute rule, for knowing how well features follow the goals of the web. Just because an idea doesn't follow REST does not mean that it is a terrible one.

A Word on REST and APIs

The word REST has come to mean so much more since 2000. In 2008, Fielding wrote "I am getting frustrated by the number of people calling any HTTP-based interface a REST API. Today's example is [..] That is RPC. It screams RPC. There is so much coupling on display that it should be given an X rating." [18]. Engineers didn't suddenly stop using the term after his proclamation though - in fact, it seems quite the opposite. Often you'll still hear engineers talking about which HTTP verb to be using with some request from a client to a server, despite this not even being mentioned tangentially in Fielding's dissertation. It is clear that the term REST was misappropriated - and it is helpful to separate the two conceptually.

Conclusion

REST was born from massive pressure driving new innovation to the web. As part of the standardization process, Roy Fielding was called on to help write specifications, working with many other smart engineers, each on their own specifications, that together would form the modern web. REST was designed as a framework to design and evaluate proposed improvements to key communication protocols. This was done by using a style (REST) as a type of acid test to prove proposals before they were deployed. The codification of the styles that induced desirable properties allowed the web to transition into what was, in their time, known as the "modern web". REST was a critical piece in a vulnerable juncture in the web's development. The principles of REST still hold up to this day–and are so solid that they have been borrowed by engineers to describe things outside of the original domain that Fielding created it for.

Acknowledgment

We thank Alex Peterson for reviewing this paper.

REFERENCES

1. R. Fielding, "Representational state transfer," *Architectural Styles and the Design of Netowork-based Software Architecture*, pp. 76–85, 2000.
2. D. M. Ritchie and K. Thompson, "The unix time-sharing system," *Bell System Technical Journal*, vol. 57, no. 6, pp. 1905–1929, 1978.
3. M. V. Wilkes, "Slave memories and dynamic storage allocation," *IEEE Transactions on Electronic Computers*, no. 2, pp. 270–271, 1965.
4. E. B. Shapiro, "Network timetable," *RFC*, vol. 4, pp. 1–6, 1969.
5. J. Rulifson, "Decode encode language (DEL)," *RFC*, vol. 5, pp. 1–17, 1969.
6. A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding code mobility," *IEEE Transactions on software engineering*, vol. 24, no. 5, pp. 342–361, 1998.
7. T. Berners-Lee, "Www: Past, present, and future," *Computer*, vol. 29, no. 10, pp. 69–77, 1996.
8. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, "Rfc2068: Hypertext transfer protocol–http/1.1," 1997.
9. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Rfc2616: Hypertext transfer protocol–http/1.1," tech. rep., 1999.
10. S. Target, Jun 2020.

11. A. Wolman, G. Voelker, N. Sharma, N. Cardwell, M. Brown, T. Landray, D. Pinnel, A. Karlin, and H. Levy, "Organiz at ion-basedanalysisof web-objectsharingandcachi ng," 1999.
12. D. D. Clark and D. L. Tennenhouse, "Architectural considerations for a new generation of protocols," *ACM SIGCOMM Computer Communication Review*, vol. 20, no. 4, pp. 200–208, 1990.
13. S. Glassman, "A caching relay for the world wide web," *Computer Networks and ISDN systems*, vol. 27, no. 2, pp. 165–173, 1994.
14. A. Luotonen and K. Altis, "World-wide web proxies," *Computer Networks and ISDN systems*, vol. 27, no. 2, pp. 147–154, 1994.
15. T. Berners-Lee, R. Cailliau, C. Barker, and J.-F. Groff, "Design notes," Nov 1992.
16. T. Berners-Lee, R. Cailliau, J.-F. Groff, and B. Pollermann, "World-wide web: the information universe," *Internet Research*, 1992.
17. T. Berners-Lee and R. Cailliau, "World-wide web," in *Proceedings of Computing in High Energy Physics 92*, (Annecy, France), pp. 23–27, Sep 1992.
18. R. T. Fielding, Oct 2008.

Matt Pope is a graduate student at Brigham Young University. He received his BS in Computer Science in 2018. His research focuses on fault tolerance in software systems, specifically how systems can degrade gracefully.

A Short History of Encryption

Garrett Smith
Brigham Young University

***Abstract*—Encryption forms the backbone for secure communication in the modern day. In ancient and modern history, encryption techniques were used to obscure trade secrets, and assist in sharing military communication. Today, we use encryption every day as we browse the web, communicate over secure messaging platforms, and access online banking and shopping. In this paper, we investigate the evolution of encryption from Ancient Mesopotamia to the public-key cryptosystems of the modern day. We explore the uses of modern encryption in cloud computing and secure messaging, and look to the future implications of the democratization of cryptography.**

■ **SINCE THE DAWN OF HUMANITY**, there has been a need for people to conceal information. There have been many different methods that people have used to protect information from being seen or understood by those whom the information was not meant for. In ancient times, this was primarily used for secret communication during war times between leaders. Even before that, one of the earliest forms of concealing information was to protect a trade secret. Today, we use newer methods to protect private health data, messages, and financial information when we use the internet. While the uses for concealing data have varied throughout history, many important principles remain the same. To understand these principles we need to understand how information is concealed, how these methods have evolved over time, and finally understand its role in our lives today and how it may change tomorrow.

Cryptography Basics

Cryptography is the study of secret communications, especially when communication may be in the presence of adversaries [1]. The primary goal of cryptography is privacy. If two individuals share information privately, an adversary would not be able to understand what was communicated. At first, these communications were typically written words, which had been disguised so that adversaries could not understand what the messages said. This is called encryption. Encryption is "the process of converting information into a code, especially to prevent unauthorized access" [2]. Rivest explains a standard encryption algorithms uses a secret-key ecosystem which contains the following:

- A message space M: the set of strings that make up a message over an alphabet
- A ciphertext space C: the set of strings that make up the ciphertexts over an alphabet
- A key space K: the set of strings that make up the keys over an alphabet
- An encryption algorithm E that maps $K * M$ to C
- A decryption algorithm D that maps $K * C$ to M

The algorithms E and D must satisfy the property that D(k, E(k,m)) = m for every k $\in K$, and m $\in M$ [1]. This means that for any key K, and message M, the encryption with E will be the ciphertext C. The algorithm D must then decrypt C using key K into M. For this system to work, the key K must be transmitted to each party out of band, meaning the key must be agreed upon by both parties privately, and separately from when they share ciphertexts. This key must be kept secret.

In this paper we will use several pseudonyms for individuals that may be communicating with each other to help illustrate how these encryption techniques work. Alice and Bob are individuals that are attempting to share information, while Eve is an eavesdropper on their communications. As long as Alice and Bob use a secure algorithm for E, and their key is kept secret from Eve, then this method provides confidentiality, the primary goal of encryption. However, today, encryption is used for other purposes as well, specifically authentication, integrity, and non-repudiation. Authentication helps Alice determine that she is communicating with Bob and not someone else pretending to be Bob. Integrity, allows Bob to detect if the message that was sent from Alice has been modified by someone who is not authorized. Non-repudiation, prevents the ability for Alice to later deny that the message was written by her. These properties that may be provided by encryption are more recent innovations in cryptography. Before we can understand the implications of these encryption purposes we must understand how cryptography has evolved over the last 2 millennia.

Before the Information Age

Today, encryption is primarily understood to be the protecting of digital information from attackers or online adversaries. However, encryption began far before the invention of the computer or even the discovery of electricity. In this section we will investigate the roots of cryptography from ancient Mesopotamia to their use in World War I and World War II.

Ancient History

Scytale The first documented example of cryptography is in Ancient Mesopotamia, near the Tigris River. A cuneiform tablet was discovered, dating back to 1500 BC which contained an encrypted recipe for a pottery glaze [3]. The author used a simple substitution process, substituting symbols for other symbols, making the writing seem random. Those trained in the method of encryption were able to decipher it allowing craftsmen to share the recipe with allies without having to worry about adversaries learning their trade secret.

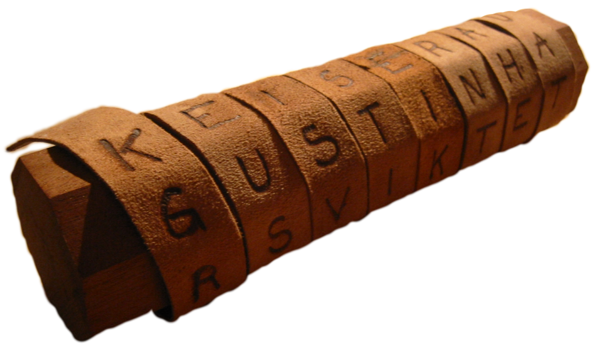


Figure 1. A representation of an Ancient Greek Scytale [5]

This encryption method is weak, and slow, especially if there is a long list of paired symbols that must be memorized. For this application the solution was probably enough, however when it came to more complicated messages, this algorithm had significant weaknesses.

Nearly 800 years later in ancient Greece, military leaders recorded the use of a device called a scytale [4]. This device was a simple rod of a specific diameter. Alice would wind a strip of parchment around the rod going from left to right, and then write a message from left to right across the strip of paper as seen in figure 1. When the parchment is removed it would look like just a series of random letters. When Bob receives the parchment he would wind the parchment around his rod of the same diameter and would be able to decrypt the message. If Eve was able to intercept the message she would need a rod of the same diameter to be able to decipher the message. For this cipher the "key" used would be the diameter of the rod, without this diameter the message would not be able to be decoded.

This process had significant advantages to the method of encryption used in Ancient Mesopotamia. It required no memorization, and was fast to both encrypt and decrypt. However it would have been fairly simple for adversaries to decrypt the messages if one knew the process. They could simply use various sizes of rods, and use different sizes until the letters on the parchments aligned into sentences and words that were sensible.

Caesar Cipher A stronger method of encryption is found several centuries later in ancient Rome during the reign of Julius Caesar, named the Caesar Cipher [6]. This cipher is one of the most well known of the basic ciphers. The cipher works by shifting each symbol in the message by a given number to mask the real symbols used. Just like the diameter of the Scytale being the secret "key", the number of the shift would be the secret "key". For example, if the number to shift is 3, the standard use by Julius Caesar, an A would become a D when encrypting. The system would wrap around the alphabet meaning a Z would become a B when encrypted. This process is simply reversed when decrypting, B would become a Z, and D would become an A.

At it's time this method was believed to be a reasonably strong method for encrypting messages. Caesar used it to send messages throughout his military. Religious leaders used it to encrypt the names of God in religious texts [7]. Even today this cipher has been found to be used by some terrorist organizations [8]. The widespread use of this cipher shows it's impact and success in protecting private messages. However, 700 years after it's first reported usage, a method for breaking this cipher was developed by Arabic cryptographers, contemporaries of Muhammad ibn Musa al-Khwarizmi, the father of algebra. The first important breakthrough at this time was by Al-Khalil ibn Ahmad al-Farahidi, born in 718 CE. Al-Farahidi was made the first dictionary of the Arabic language. Following this work, he wrote one of the first books focused on cryptography and cryptanalysis. In this work, he listed permutations of all possible Arabic word [9]. Following on this work Abu Yusuf Ya'qub ibn 'Ishaq as-Sabbah al-Kindi developed a method for breaking ciphers by analyzing the frequency of each letters occurrence in plaintext. This method is known as frequency analysis [10].

To break the Caesar cipher using frequency analysis one would first need to be familiar with the language of the plaintext message. One could analyze any plaintext message in that language of a reasonable length by counting the occurrences of each letter. This process is repeated in the ciphertext. Then, by comparing the two distributions, a pairing could be created, pairing letters of similar distributions between the plaintext and the ciphertext. Substituting each paired letter in the plaintext distribution for the accompanying letter in the ciphertext should yield the plaintext message. This method for breaking the Caesar cipher is also able to break any encryption algorithm that relies on simple substitution. Future cipher designs were primarily motivated by this development.

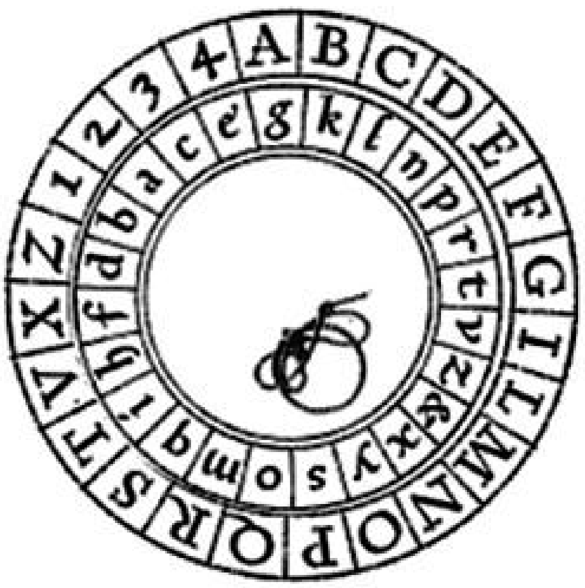

1 2 3 4 A B C D E F G I L M N O P Q R S T V X Z
a c e g k l n p r t v z & x y s o m q i h f d b

Figure 2. An Alberti Cipher Disk. [11]

Early Modern History

Alberti Cipher With the advent of frequency analysis, in 1467 Leon Battista Alberti sought to find a cipher that would be immune to these attacks. In the process he developed the first polyalphabetic substitution cipher. A polyalphabetic substitution cipher is a cipher that switches between multiple alphabets when encrypting a message. The cipher he invented is known today as the Alberti Cipher.

The Alberti Cipher uses an Alberti Cipher Disk. This device consists of two concentric disks which can be rotated with respect to each other, seen in figure 2. Each disk was divided into 24 equal sized sections with each section containing a unique symbol. On the other ring were the 20 capital latin letters minus the letters H, K, or Y and the numbers one through four. On the inner disk were the lowercase latin letters with the & symbol included in a random order.

Alberti outlined multiple ways to use this cipher. For our purposes we will outline the method primarily used. For this method both Alice and Bob would need to have an Alberti Cipher Disk and would have to have agreed beforehand to use one of the lowercase letters as their chosen index. When Alice is encrypting a message, she would first choose a random capital letter and position their agreed upon index beneath that capital letter. She would then write that capital letter in the cipher text followed by the symbol in the inner ring that aligns with the location of the first letter of the plaintext on the outer ring. Alice would then move to the next characters in the plain-text, writing the corresponding cipher text. This would be repeated for a few words before Alice would then shift the cipher by rotating the inner ring some number of spaces then writing the capital letter that is above the index in the cipher text and then repeating the process until the message is completely encoded. To decode the message Bob would start at the beginning on the ciphertext and put the index of his disk on the first letter in the message. Then writing the corresponding outer symbol for each letter in the ciphertext, switching the location of the index everytime a capital letter or number appears in the ciphertext.

At its time, this method seemed to be immune to frequency analysis attacks, however, future worked proved that this cipher still suffered from vulnerabilities. The first problem is that the cipher relies on the agreed upon method for shifting the alphabet used for encrypting to be secret. If Eve is able to intercept a message sent from Alice and knows that the capital letters represent a shift of the index to that letter, she just needs to figure out what index Alice used. Since there are only 24 options for the index value she could simply iterate through those 24 options until the first few words are successfully decoded. Every time a new capital letter appears Eve could just shift the index she found to match that capital. This is known as a brute force attack. Since there are only 24 options, this attack is fairly easy to implement. Despite it's weaknesses this development helped pave the way for the modern encryption methods of today.

Vigenere Cipher One hundred years later, in 1553, Giovan Battista Bellaso created a new polyalphabetic cipher that would improve on the Alberti Cipher and even be termed unbreakable for almost 300 years [12]. Despite Bellaso being the inventor, in the 1800's the cipher was misattributed to Blaise de Vigenere and was thus named the Vigenere Cipher [13]. Bellaso built this cipher on prior work from Alberti, as well as the work of Johannes Trithemius, who invented the *tabula recta*. The *tabula recta* was a letter square as seen in figure 3 which can be understood to be a table of 26 different Caesar ciphers. The first row represents the plaintext alphabet, and each subsequent row is the row previous being shifted by one letter. Bellaso recognized that he could use this table to shift each letter by a varying amount by using a secret key.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Figure 3. A Vigenere Square, or Tabula recta. [14]

If Alice used the Vigenere Cipher to encrypt a message for Bob they would first need to decide on a secret key, typically a word or phrase. Say, they decide on the key being "PASSWORD." Now Alice needs to encrypt the message "We attack at dawn." Then Alice would align the plaintext with the key as seen below. Then using the *tabula recta* Alice would look down the column that starts with "W" and select the letter that cooresponds with the row that starts with "P", the first letter in the key and would find the letter "L." She would then repeat that process for each letter in the message, the resulting string would be the cipher text. Bob would then "undo" this by following a similar method, except instead of finding the correct letter in the table, instead Bob would start with the row that starts with the first letter of the password, looking at each column until he see's the first letter in the ciphertext. The column that the cipher text letter is in would be the plaintext letter.

$$WEATTACKATDAWN$$

$$PASSWORDPASSWO$$

$$LESLPOTNPTVSSB$$

Mathematically, each of these letters can be assigned to the numbers 0-25, where A is 0, B is 1, etc. Alice would have a message M, a string of letters M_i , and some key K, a string of letters K_i. The resulting cipher text C would be:

$$C_i = (M_i + K_i) mod 26$$

The decryption d would then be,

$$M_i = (C_i - K_i + 26) mod 26$$

This encryption algorithm improved on Alberti's, significantly increasing the complexity of the encryption and increasing the difficulty to crack. Despite claims that it was unbreakeable, in 1863 Friedrich Kasiski published an attack on the Vigenere cipher. The key weakness in the cipher was identified to be the length of the key that was used. While under most circumstances it was impossible to use frequency analysis to crack the cipher, Kasiski found that if there were repeated strings in the message, there would be times that the repeated words would be encrypted to the same thing, if the words aligned with the length of the password. Once an attacker can identify the length of the key used, they can then break the cipher text into blocks the size of the key used. In each block the nth letters will have been encrypted using the same alphabet because they were encrypted with the same letter in the key. By doing frequency analysis on the nth letters combined. If the message is long enough for frequency analysis this process will reveal the key that was used, thus breaking the cipher.

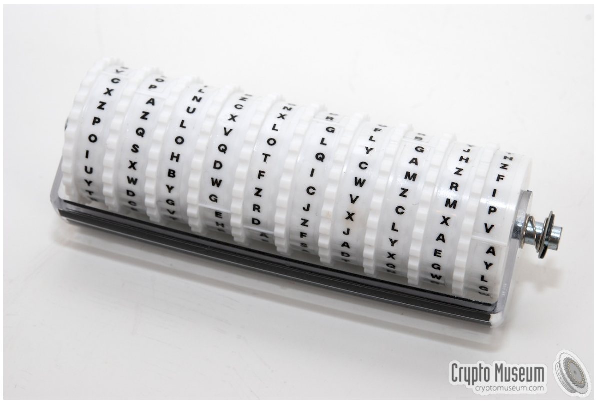


Figure 4. A toy replica of a Jefferson Wheel.[15]

Modern History

Now we turn our attention to the Industrial Era. With the rise of major global powers, came the need for stronger encryption, especially after the attack on the Vigenere cipher was published. The next important invention occurred before the Vigenere was known to be broken, by the American President Thomas Jefferson.

Jefferson Wheel The Jefferson Disk, invented in 1795, is a mechanical device that used a set of disks, each of which had 26 letters on the outside as seen in figure 4. Each disk had the letter scrambled in a random way so that each disk is different. When constructing a message the disks would be placed in an order that both parties agree upon. Then a sender could rotate each disk to spell out their message, and then choose any other row of the disk to be the cipher text. This would be repeated until the message is completely encrypted. To decrypt the receiver would put the disks in the correct order and spell out the ciphertext by rotating the disks. Then by looking at every row, they could easily find the row that is readable and that would be the plaintext.

The key in this algorithm is the order of the disks. Without any knowledge of the plaintext or the key, the only way to crack the message is by trying out every possible order of disks. Jefferson's original design had 36 disks meaning there would be $36! \approx 2^{138}$ possible orderings, an unreasonable amount of work even for modern day computers. On the other hand, if an attacker knew some information about the message such as the first block of the plaintext, they could use this to determine the offsets used between the plaintext and the ciphertext to find the correct ordering of the disks. A similar method for attacking the Jefferson wheel enabled Polish and British cryptographers to attack one of the most infamous encryption devices in history, the German Enigma.

Enigma Machine The German Enigma machine, was a device used by the German military throughout World War 2. During WWII, the Germans adopted a military fighting tactic they called Blitzkrieg, which emphasized a rapid, overwhelming force oftentimes in surprise. In order to achieve this, the German military relied on radio signals to instantly communicate with military leaders but they needed a way to keep the messages from being understood by their enemies. To do this they encrypted their messages using these Enigma machines. The machine was a rotor machine that could scramble the 26 letters of the alphabet. When encrypting a message the sender would type the message on the keyboard and when each key is pressed one of 26 lights would be pressed which would correspond to the ciphertext. The encrypted message could then be sent over radio waves without fear that the message would be decrypted.

The machine also used a polyalphabetic cipher and builds upon the Vigenere Cipher and Jefferson's wheel. Instead of using a specific text key like in Vigenere the key of the system was on how each machine was configured. Each machine had specific rotor settings and wiring settings. These settings determined how each rotor would rotate with each key press of the keyboard. Like the Jefferson wheel, the possible permutations for the settings made a brute force attack nearly impossible.

In September of 1939 in the middle of WWII, up and coming British mathematician Alan Turing joined cryptographers at Bletchley Park and began work on breaking the Enigma. As mentioned earlier, because the vast number of possible permutations of the settings for the Enigma prevented a brute force attack, they had to look for other methods of attack. In the process they found that messages sent that were a continuation of a previous message always started with the same plaintext. Using this, researchers were able to store every permutation they had seen of the ciphertext, and they could then systematically determine the corresponding machine configurations. This process still required intense computational effort, and to do this Turing invented the Bombe. This device would replicate the action of many enigma machines wired together and could reduce the space of possible permutations. Turing's work was successful, and the British were able to crack the Enigma. By many accounts their work led to the shortening of the war and a British victory [16].

One Time Pad After finishing work at Bletchley Park, Turing traveled to America to assist the American efforts in breaking the Ciphers used by the German Navy in the North Atlantic. While there he came into contact with Claude Shannon at Bell Labs. Shortly after the war Shannon worked on developing methods for proving the security of encryption techniques. In the process he proved the security of one-time pads, an important encryption technique invented in 1882 by Frank Miller [17].

The one time pad is very similar to the Vigenere cipher, except for a few major differences. In a one time pad, the key must be at least as long as the message that is to be encrypted. When a sender wants to sent an encrpyted message they would do modular addition of the plaintext message with the key. As long as the key is completely random, not reused, and the length of the plaintext it is impossible to crack. An example can prove this case. Let's say Alice needs to send the message "ONETIMEPAD" to Bob. They would first need to have agreed to a secret key that is at least 10 characters long. If they randomly created the key "TBFRGFARFM" Alice could then encrypt the plaintext as shown below.

$$ONETIMEPAD$$

$$TBFRGFARFM$$

$$IPKLPSFHGO$$

If Eve was able to intercept the ciphertext she could attempt to iterate through possible keys but there would be a significant number of viable options. For example, she could find the key "POYYAEAAZX" would decrypt the message to "SALMONEGGS" or "GREENFLUID." Each of these are equally possible and therefore it would be impossible for Eve to learn the correct plaintext [18]. Shannon realized this fact and published his proof showing how the one time pad is perfectly secure [19]. This proof became a critical bedrock for modern encryption. Shannon helped illustrate the connection between mathematics and cryptography and described the two basic types of cryptography. He explains one form of cryptography is that like the one time pad, a system meant to protect against adversaries with infinite resources, and practical secrecy, being systems meant to protect against adversaries with finite resources [19]. Shannon's work served as a major influence for future cryptographic advancements in the Information Age.

The Information Age

We will now shift our attention to the two primary breakthroughs in the Information Age that followed the groundbreaking work of Shannon. With the rise of computers and electronic communication, came the need for a standard method of protecting digital communications. The primary focus since Shannon's proof has been on practical security systems.

Symmetric Encryption

In the 1970's the National Bureau of Standards invited researchers to developed a technique for securing communications for large financial organizations. In response to this request IBM presented their proposed solution based on a design by Horst Feistel.

Feistel Networks As a researcher at IBM, Feistel designed the Feistel Network . The Feistel network is a structure that can be used in the implementation of block ciphers. A block cipher is simply a cipher that uses a key and operates on a fixed size of data called a block. Typically block ciphers use a function called a round function that takes a key and some data as an input and returns an output the same size as the input data. This round function is repeated some number of times as defined by the specific algorithm. In most

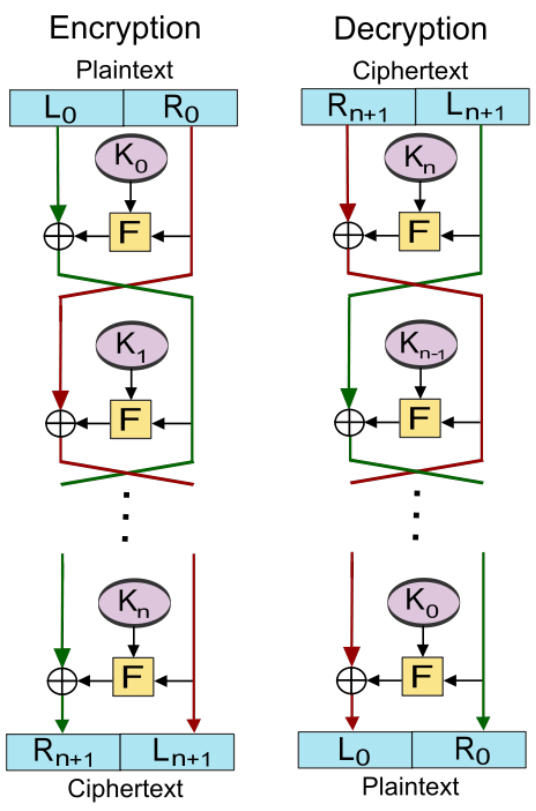


Figure 5. The flow diagram for encryption and decryption for a Feistel Network. [21]

circumstances a master key of fixed size is shared between parties and the encryption algorithm uses a key schedule to derive n number of subkeys where n is equal to the number of rounds that the round function is repeated. Feistel Networks follow this basic operation, seen in Figure 5 . First a block is divided into two blocks of equal size, L_0 and R_0. One block, R_0 is used as an input to a round function with a subkey derived from the master key. The resulting output, is XORed with the block L_0 to get L_1. In the next round L_1 becomes the input to the round function and it's result is XORed with R_0 to get R_1. This process is then repeated n times [20]. One important characteristic of Feistel networks is that the method for decryption is the exact same except for the process for calculating the subkeys are reversed. Because the round function is not reversed, the function does not need to be invertible.

The proposed solution by IBM was accepted by the National Bureau of Statistics and named the Data Encryption Standard (DES) with a key size of 56-bits [22]. This standard was used until it was superseded by a new encryption standard, the Advanced Encryption Standard in 2001.

Substitution-Permutation Networks - Rijndael - AES In the late 1990's researchers began to be concerned with the small key size used in DES [23]. In 1997, the National Institute of Standards and Technology, the renamed National Bureau of Standards requested a new standard for data encryption, this time holding a competition among interested researchers. In all, 15 algorithms were proposed with the eventual winner being the Rijndael algorithm, named for the two authors Vincent Rijmen, and Joan Daemon, renamed the Advanced Encryption Standard (AES).

AES uses a different type of block cipher than DES, a substitution-permutation network (SPN). SPNs like Feistel networks apply some defined operations over a series of rounds. However, instead of using a function like the round functions in Feistel Networks, SPNs use a combination of substitution boxes (S-Box) and permutation boxes (P-Box) as seen in Figure 6. The S-Boxes perform a substitution of the bytes in the input and produce a different output. This must be one to one so that the S-Box is invertible. In the P-Box the input is simply scrambled in a specific order and then some operation, usually an XOR, is performed with a subkey. This is then repeated over a given of n rounds. Unlike Feistel Networks each of the S-Boxes and P-Boxes must be invertible. To decrypt the ciphertext, the algorithm is reversed using the inverse S-Boxes and P-Boxes [24].

The accepted version of AES originally used a key size of 128. Since then, with advancing computing power, the key size has been increased to 256 bit key sizes.

Public-Key Cryptography

One of the most significant advancements in the world of crpytography was the invention of asymmetric encryption or Public-key cryptography. In each cryptopgrphic algorithm discussed today, there is one requirement that is consistent, the need for communicating parties to agree on a secret key and be able to share this key securely with each other. This problem had plagued cryptographic methods for centuries. In the 1970's the problem was especially on the mind of Whitfield Diffie, an American Computer Scientist. While trying to think of ways to improve key distribution he imagined a groundbreaking scenario in which encryption and decryption could be asymmetric instead of symmetric. This process would allow each party to have two keys, a public key used for encrypting and a private key used for decrypting. For Alice to send a message to Bob, she would encrypt the message with Bob's public key which Bob could share over an insecure channel as seen in Figure 7. When Bob receives the message he could then decrypt the message using his private key which he would keep secret. The important aspect of this is that decryption can only be done with the secret private key, so an eavesdropper could not decrypt any intercepted messages. Diffie worked together with Martin Hellman to invent this idea, however they had no mathematical way to construct such a system.

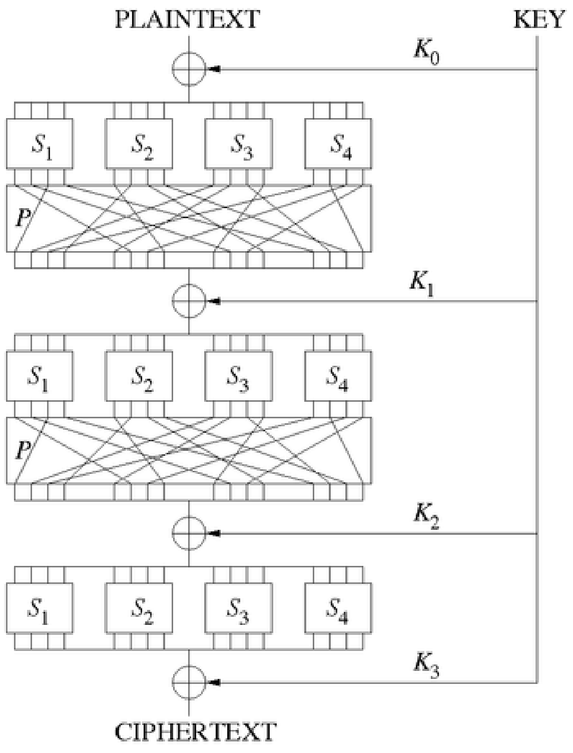


Figure 6. The flow diagram for a Substitution-Permutation network. [25]

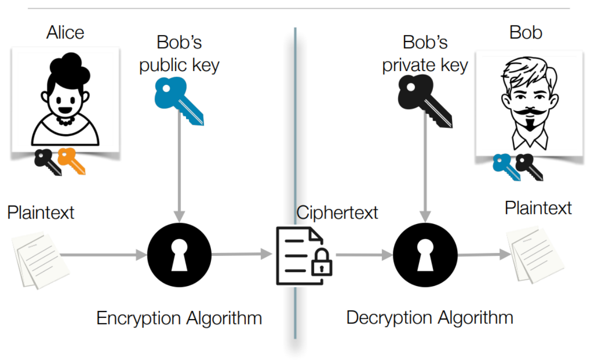


Figure 7. A basic flow diagram for a public-key cryptographic system [26]. This example shows what keys are used when Alice encrypts a message to Bob, and Bob unencrypts the message.

Merkle Puzzles While considering this problem in 1974 Ralph Merkle a PhD student of Martin Hellman, published a protocol in which two individuals could communicate over an insecure channel and negotiate a shared secret without an eavesdropper being able to learn the secret key. The protocol involved Alice generating a series of puzzles which were simply encrypted messages that when unencrypted would reveal an identifier and a session key. Each puzzle would be encrypted with a different key short enough that a brute force attack is possible and each identifier and session key would be different. Alice would send all of these puzzles to Bob. Bob would then randomly select one of the puzzles and use a brute force attack to get an identifier and session key. Bob could then tell Alice the identifier and they could then encrypt their messages back and forth with each other using this session key. While the process is simple for Alice and Bob, an attack by an Eavesdropper is more difficult. Eve couldn't simply crack one of the puzzles, she would need to statistically solve half of them before learning the session key [27]. While this protocol is not computationally complex enough for practical usage it opened the door for the study of what are called trapdoor functions, functions that are computationally easy to perform in one direction but very difficult to perform in reverse. It is this discovery that led to the real world systems that are still in use today.

Diffie-Helman Shortly after Merkle's invention, Hellman and Whitfield Diffie published their proposed public-key cryptography system named Diffie-Hellman key exchange and originated the idea for a trapdoor function [28]. The protocol involves the parties publicly choosing a large prime number p, and a primitive root, base g. This means that g is an integer less than p such that for every number n from 0 to p there is a power k such that $n = g^k modp$. Both parties would then generate some random private integer value a, or b. They would then calculate a public value $g^a modp$ or $g^b modp$ and share these values publicly. After doing this they can calculate a shared secret $g^{ba} = (g^b)^a modp = (g^a)^b modp = g^{ab}$. An eavesdropper can not directly calculate g^{ba} unless they know either a or b. Since they do not know either they have to attempt to solve the discrete logarithm problem which is a very difficult problem for large numbers.

This system is secure against a passive attacker, or eavesdropper, however it can be attacked by a Man-In-The-Middle, because there isn't a way for the two parties to authenticate to each other. While Diffie and Hellman were working on this protocol, three other researchers were simultaneuously working on their own solution, one which satisfied the idea's first proposed by Diffie.

RSA After Diffie and Hellman published their findings, computer scientists Ron Rivest and Adi Shamir with mathematician Leonard Adleman began looking for possible candidate one-way functions that would be difficult to invert, similar to Diffie-Helman trapdoor problems. The story goes that Rivest, Shamir, and Adleman spent Passover at a students home and apparently consumed a significant amount of Manischewitz wine before returning to their homes late that night. Rivest returned unable to sleep, and lay on his couch near his math textbook. While in this state, he discovered a function that could satisfy their problem. He spent the rest of night formalizing the idea, by the next morning the mathematical paper was nearly finished [29]. Together they proposed their system Rivest-Shamir-Adleman or RSA.

RSA involves 4 steps. First, users needed to generate a private-public key pair. Second, they needed to share their public key over a reliable but not necessarily secure channel. Next, using modular exponentiation encryption and decryption are possible. The protocol is as follows [30]:

Key Generation

1) Choose two random prime numbers p, and q.
2) Calculate $n = pg$.
3) Choose an encryption key e such that e and $(p-1)(q-1)$ are coprime.
4) Use the extended Euclidean algorithm to compute the decryption key d
$$d = e^{-1} mod((p-1)(q-1))$$
5) Finally, d becomes the private key and e and n become the public key.

Encryption and Decryption

1) The cipher text is calculated using
$$c = m^e modn$$
2) The plain text can be calculated by
$$m = c^d modn$$

This algorithm security relies upon the assumption that factoring large numbers is a difficult problem to solve [18]. If an attacker could easily factor N into p and q. While no mathematical proof can show this difficulty, as of today the largest publicly known factored RSA number is 829 bits which took 2700 CPU years to compute [31]. RSA keys are typically between 1024, and 4096 bits long. These two protocols form the bedrock for modern-day communication systems such as the Transport Layer Security (TLS) used in encrypting internet communications and secure messaging applications like Signal.

The Future of Encryption

With this evolution of cryptography we reach our modern age of cryptography. Since Shannon's work on proving the secrecy of systems, we've developed systems that enable all of humanity easy access to strong methods of encryption. This is not without significant downsides. While these strong methods protect users' sensitive information from hackers, or despotic regimes, it also allows criminals and terrorists to communicate securely with each other without fear of law enforcement being able to intercept and read their messages.

Encryption in Politics

Naturally, as these strong methods of encryption were published there was a push by governments to restrict access to these tools. When DES was first chosen as the standard encrpytion technique, the NSA first had to approve the device in which they required the key size to be reduced presumably because they had the capability of doing a brute force attack on that key size [32]. Throughout the Cold War and through he 1990's the US government implemented strict control especially on the exporting of cryptographic algorithms as they were determined to be non-exportable weapons. After years of lawsuits and public complaint, at the end of the 90's most of these restrictions were lifted and secure cryptography was adopted worldwide.

Despite this success, even today the arguments against uncrackable encrpytion thrive in the public sphere. As internet connectivity expanded, and the rise of the dark web, government authorities have been hindered in their ability to stop the proliferation of child pornography. With this issue, the arguments against encryption rose again, with the Attorney General of the United States Bill Barr working with a Republican congress to enact the Eliminating Abusive and Rampant Neglect of Interactive Technologies Act of 2020 or EARN IT Act. This act would require any providers of end to end encryption provide back-door access to the government, despite most security researchers being opposed to such requirements [33]. These experts believe primarily that creating a backdoor for end to end encryption weakens the system altogether, creating another vector for attack. Further, with tech companies desire to expand their market share into countries outside of the US, it is likely that if a backdoor is created for the US, countries such as China could also coerce these companies to provide this backdoor access for them in exchange for access to the Chinese market. Whether this legislation is enacted, remains to be seen.

Secure Messaging

Just as encryption allows evil to be perpetuated online, it also protects activists, dissidents, and journalists. Recently, secure messaging ecosystems such as WhatsApp, and Signal have seen widespread adoption. These apps allow users to communicate with each other with their message being end to end encrypted. This allows dissidents and activists to communicate sensitive information with journalists and each other. However, this has created the issue that users of these apps could become targets. In China, many individuals have reported being targeted by the Chinese regime simply because they had secure messaging apps on their phone [34]. For many, the benefits provided outweigh these risks when living in a surveillance state.

Homomorphic Encryption

One of the most interesting work being done in cryptography currently is homomorphic encryption. Homomorphic Encryption is simply an encryption scheme that allow computations to be performed on the ciphertext of some encrypted data without decrypting the data first. Most systems such as AES do not have this property, but there is a significant interest in developing a system that would support this. It's applications can be far-reaching, however the primary interestd parties are highly regulated industries such as the health care industry. Homomorphic encryption would allow these industries to outsource their data storage and computation to third-parties such as cloud computing services, while simultaneously protecting their clients data. Google. IBM, and others have proposed some types of cryptosystems that would allow some forms of calculations but there has yet to be a system created that satisfies the necessary requirements.

Quantum Computing

Today, we have an incredible amount of computational power. These advances in technology have helped us to achieve the necessary computational resources needed for strong cryptography to exist. However as technology evolves, there is the worry among many that quantum computing could make our modern cryptography obsolete. In response to this worry, NIST has accepted proposals for a quantum-resistant algorithms to be standardized. Nearly 70 proposals were made and the selection process has reduced the number to 15. It is believed that a standard could be implemented before quantum computing becomes a real threat.

CONCLUSION

Throughout history encryption has primarily been used by governments, and militaries to protect government secrets and provide strategic advantage. With the advent of the digital age, we observe the great democratization of encryption. Instead of only being used in rare circumstances, today's society uses cryptographic algorithms every single day, whether they recognize it or not. Understanding the evolution of encryption allows us to understand the weaknesses in our modern methods, and a systematic method for categorizing the security of encryption techniques.

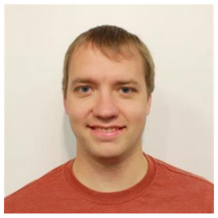
ACKNOWLEDGMENT

I thank Casey Sun, for their incredible patience and understanding in reviewing this. Their comments helped improve this paper. We also thank our Editor Michael DeBuse for his assistance in the editing process.

■ REFERENCES

1. R. L. Rivest, "Cryptography," in *Algorithms and complexity*, pp. 717–755, Elsevier, 1990.
2. 2020.
3. J. J. Tattersall, *Elementary number theory in nine chapters*. Cambridge University Press, 2005.
4. M. Djekic, "A scytale–cryptography of the ancient sparta," 2013.
5. Wikipedia, the free encyclopedia, "Scytale," 2007. [Online; accessed December 11, 2020] https://commons.wikimedia.org/wiki/File:Skytale.png.
6. S. Singh, *The code book*, vol. 7. Doubleday New York, 1999.
7. "Mezuzah and astrology."
8. "Ba jihadist relied on jesus-era encryption."
9. L. D. Broemeling, "An account of early statistical inference in arab cryptology," *The American Statistician*, vol. 65, no. 4, pp. 255–257, 2011.
10. M. Cozzens and S. J. Miller, *The Mathematics of Encryption*, vol. 29. American Mathematical Soc., 2013.
11. Buonafalce, Augusto, "Alberti cipher," 2008. [Online; accessed December 11, 2020] https://en.wikipedia.org/wiki/Alberti_cipher#/media/File:Alberti_cipher_disk.JPG.
12. B. Preneel and V. Rijmen, *State of the Art in Applied Cryptography: Course on Computer Security and Industrial Cryptography, Leuven, Belgium, June 3-6, 1997 Revised Lectures*. Springer, 2003.

13. M. Coles and R. Landrum, “Introduction to encryption,” in *Expert SQL Server 2008 Encryption*, pp. 1–20, Springer, 2009.
14. Fields, Brandon T., “The vigenère square or vigenère table,” 2011. [Online; accessed December 11, 2020] https://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher#/media/File:Vigen%C3%A8re_square_shading.svg.
15. Crypto Museum, “Jefferson disk,” 2010. [Online; accessed December 11, 2020] https://www.cryptomuseum.com/crypto/usa/jefferson/index.htm.
16. J. J. Gunnarson, ““world war ll was shortened by at least two years “(bletchley park),” *World War II Comes to the Northern Plains*, p. 94, 2016.
17. F. Miller, *Telegraphic code to insure privacy and secrecy in the transmission of telegrams.* CM Cornwell, 1882.
18. B. Schneier, *Applied cryptography: protocols, algorithms, and source code in C.* john wiley & sons, 2007.
19. C. E. Shannon, “Communication theory of secrecy systems,” *The Bell system technical journal*, vol. 28, no. 4, pp. 656–715, 1949.
20. J. Katz and Y. Lindell, *Introduction to modern cryptography.* CRC press, 2014.
21. Amirki, “Feistel cipher,” 2011. [Online; accessed December 11, 2020] https://commons.wikimedia.org/wiki/File:Feistel_cipher_diagram_en.svg.
22. F. PUB, “Data encryption standard (des),” *FIPS PUB*, pp. 3–46, 1999.
23. P. Van De Zande, “The day des died,” *SANS Institute*, 2001.
24. N. F. Pub, “197: Advanced encryption standard (aes),” *Federal information processing standards publication*, vol. 197, no. 441, p. 0311, 2001.
25. Gabor, Pete, “Substitution permution network,” 2009. [Online; accessed December 11, 2020] https://en.wikipedia.org/wiki/File:SubstitutionPermutationNetwork2.png.
26. Zappala, Daniel, “Public key cryptography,” 2019. [Online; accessed December 11, 2020] https://cs465.internet.byu.edu/static/lectures/v1/Cryptography.pdf.
27. R. C. Merkle, “Secure communications over insecure channels,” *Communications of the ACM*, vol. 21, no. 4, pp. 294–299, 1978.
28. W. Diffie and M. Hellman, “New directions in cryptography,” *IEEE transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.
29. M. Calderbank, “The rsa cryptosystem: History, algorithm, primes,” *Chicago: math. uchicago. edu*, 2007.
30. R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.
31. F. Boudot, P. Gaudry, A. Guillevic, N. Heninger, E. Thome, and P. Zimmermann, “Factorization of rsa-250, 2020,” 2020.
32. “The legacy of des - schneier on security.”
33. “The earn-it act - schneier on security.”
34. 2020. https://www.amnesty.org/en/latest/news/2018/09/china-up-to-one-million-detained/.



Garrett Smith is a graduate student at Brigham Young University. He received a B.S in Mechanical Engineering from Utah State University. Currently, he works on problems related to security and privacy, especially as they relate to usability at the Internet Security Research Lab. In his spare time he enjoys spending time with his wife and daughter, cooking, and 3d printing.

Modern Human Identification

Z. Casey Sun
Brigham Young University

***Abstract*—The evolution of identity verification technology starts from the origin of human civilization. People need to declare their exclusive ownership and privileges. Then the locks and keys are invented. We list three types of human identification methods in this article and discuss their state-of-the-art progress. These approaches coexist successfully in our modern life, as our identification needs vary in different scenarios. We also introduce some novel works that explore future identity verification methods. We also discuss the ethical problems in modern human identification.**

HUMAN IDENTIFICATION is the process of verifying the identity of people. The verified person may have access to restricted resources or some privileges. Therefore, disapproved people could abuse the trust-based rule. The approaches to identify a person are very diversified. We organize this article with the known taxonomy of identification methods, see Figure 1, and introduce the most popular ones and research trends.

The size of the human community has changed from the tribe to the global. Our ancestors, who lived in clans, recognized people using their appearances. When towns emerged, people started to use signatures or seals. Nowadays, we have the chance to meet people from anywhere in the world. Biometric technology, such as face recognition, could help users identify even people we never see. Simultaneously, the manufacturing industry is changing, metal forging is still essential, and semiconductor and integration circuits are proliferating since the 3rd industrial revolution. We have seen some novel keys in our daily life.

We will discuss the origin of some identity verification methods and introduce significant or popular ones. At the end of this article, we will look at some new techniques proposed in academics. We will also cover ethics in modern human identification.

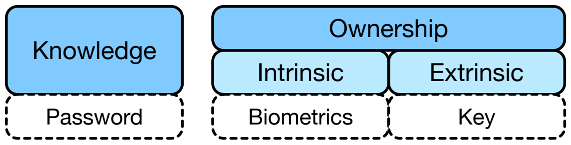


Figure 1: The taxonomy of identification methods. The examples are in the dotted box.

Keys

Locks and keys help people make themselves the only person having access to their possessions. The first key appeared about 6000 years ago, but until 200 B.C., Romans started to use forging technology to manufacture metal keys, a prototype of our modern keys. Since the industrial revolution in the late 18th century, the keys' complexity and sophistication have significantly increased. In recent years, the changes in the traditional keys usually emphasize portability and aesthetics.

We can see the disadvantages of using keys for identification:

- *Volatile*—The metal keys have a long life, but it could be lost or stolen.
- *Replicability*—Most keys are not very sophisticated. People can make a copy easily. It means more than one person could pass the identity verification.

The magnetic stripe cards[1] are one kind of modern key. In the 1950s, magnetic recording of

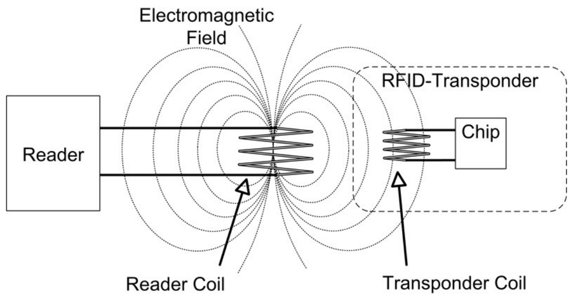


Figure 2: The RFID transponder (tag) and reader[3].

digital computer data on plastic tape coated with iron oxide was invented. Then it was possible to write and store confidential information into a card. It is much lighter than a key and able to store any binary data. The stored data is invisible to human eyes. However, this technology shares the same disadvantages as the traditional keys. The card reader can make a copy of all of the data inside. The magnetic stripe cards have been popular for a long time. Some institutes not asking for high-level security still use them. When we talked about keys, we cannot avoid the second disadvantage mentioned above, except for what I cover later in the section on "Ethics in Human Identification". However, we already solved the replicability problem in some advanced keys.

Integrated Circuit cards

Financial institutions, such as the banks, have noticed the replicability problem with magnetic stripe cards. They are promoting the Integrated Circuit (IC) cards. Each IC card has a computing system on it [2]. Only the authoritative machine can have access to the inside chip. This scheme makes it nearly impossible to reproduce an IC card without authority.

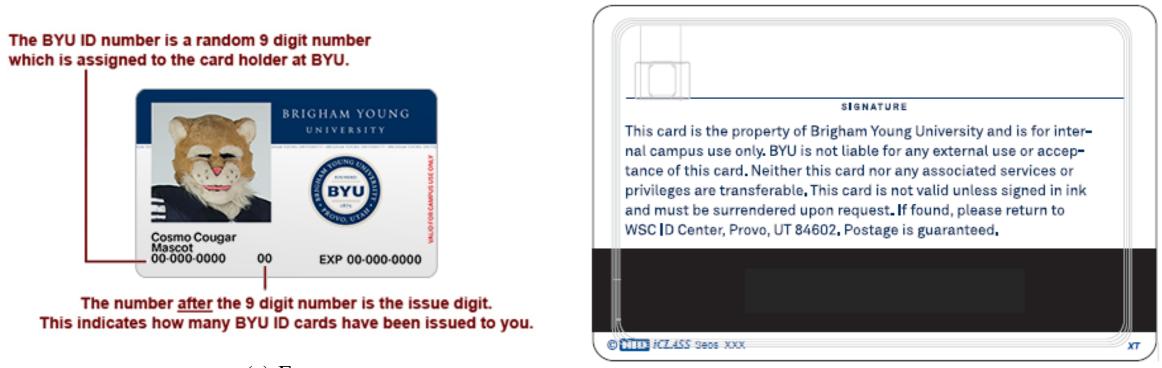
There are two kinds of IC cards. One is the contact card, and the other is the contactless card. There are eight metal contacts on the surface of the card. They are physically connected to the card reader at the verification stage to achieve power supply and complete data communication.

The contactless IC cards use radio waves for the communication between card and card reader. This technology is called Radio Frequency Identification (RFID). Its typical working frequency is 13.56 MHz. The contactless IC tag in Figure 2 is a passive RFID device. When the reader reads the card, the signal sent by the reader is composed of two parts: one is the power signal, which is received by the card, and its LC circuit generates instant energy to power the chip. The other part is the instruction and data signal, which instructs the chip to complete data reading and returns the signal to the reader. BYU ID cards also have an RFID tag inside. See Figure 3b.

Passwords

When people attempt to access a system, the password is the most commonly used authentication approach. Before the modern computer age, some mechanical locks already replace the key using a passcode. The interface is the mechanical keypad. If the keypad was in a designated state—the input digit sequence is correct, the mechanical system becomes unlocked. It is beautiful that people can update the password later by changing the inner state of the lock. The shortcoming is that the lock's total state space is usually limited; the intruder can enumerate all the solutions to unlock it. This kind of mechanical system is also a computer. There is the state concept in its design, which is the cornerstone of modern computers.

Now that the internet and computers, including laptops and smartphones, became necessities of our daily life, we need to deal with lots of passwords. The most common password we used is the keyboard input sequence. Different scenarios have different passwords rule. Some systems, such as bank cards and phone SIM cards, only require a personal identification number (PIN) code, usually a 4-digit sequence. A PIN code is easy to remember. While the system needs a physical card and limited-time tries, it is still secure. Most passwords we use in the digital world are a sequence of at least six characters, and each character chosen from the 94 possible characters using the standard US keyboard. A 8-character password potentially using the full 94 character space has over 722 quadrillion combinations. Some systems add additional rules, such as not using the common digit sequences, and must contain at least one special character, etc. We call these strong passwords. Strong passwords


The BYU ID number is a random 9 digit number which is assigned to the card holder at BYU.

The number after the 9 digit number is the issue digit. This indicates how many BYU ID cards have been issued to you.

This card is the property of Brigham Young University and is for internal campus use only. BYU is not liable for any external use or acceptance of this card. Neither this card nor any associated services or privileges are transferable. This card is not valid unless signed in ink and must be surrendered upon request. If found, please return to WSC ID Center, Provo, UT 84602. Postage is guaranteed.

(a) Front

(b) Back

Figure 3: BYU ID card design.

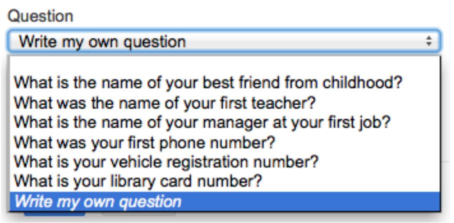
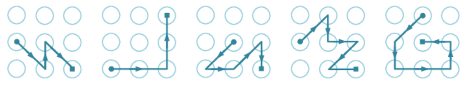
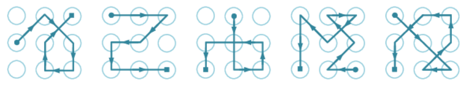

Question
Write my own question
What is the name of your best friend from childhood?
What was the name of your first teacher?
What is the name of your manager at your first job?
What was your first phone number?
What is your vehicle registration number?
What is your library card number?
Write my own question

Figure 4: The security questions[4].


(a) Example patterns belong to the simple category.


(b) Example patterns belong to the median category.

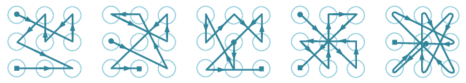

(c) Example patterns belong to the complex category.

Figure 5: The pattern examples of Android pattern lock[5].

can be very distinct but complicated and hard to memorize. If the user writes down the password, it could be discovered by someone else. Anyone possessing the correct code can act as an authorized user. Some systems use a group of passwords, which are the answers to 3 security questions, see Figure 4. The solution space is larger than the single strong passwords, but the user could answer the question using their specific life knowledge, which does not require the user's specific memory.

Another issue is that typing in a strong password on mobile devices may be inconvenient. Android devices have a solution that uses the customized pattern on the 9-node 3×3 array as the passcode, see Figure 5. The verification stage could be done using a single hand and quickly. The total number of possible unique patterns is over 300k, but people usually only use the simple ones among them. If the user only uses five nodes to design the passcode, there are about 7k patterns to choose from. The pattern lock has a similar security level to a PIN, while the 4-digit PIN code has 10k combinations. These systems usually have a limitation on the try times, and are widely popular.

Password Manager Software

One disadvantage of the passwords is that anyone knowing it can pass the identity verification. The risk is either zero or very high. The worst case is when some people use the same password for all of their private accounts. Once one of them is leaked, the hacker may try to request access to the service provided by other companies using the leaked password. The best solution is to use a password manager software, which helps the user design a distinct ultra-strong password for each private account and store it. Access to the software itself usually needs at least two stages of identity verification to ensure security.

Biometrics

In the section on keys, we discussed that fraud could happen when the ownership of the

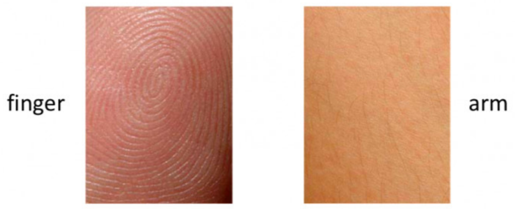


Figure 6: Two types of skin on the human body[8].

key is lost. When we use the smart ID card on the BYU campus, the servicemen need to check the profile photo printed on the card to double-check the user's identity. This procedure employs face verification, which is on type of biometrics. Each human subject has distinct genes and nurtured training. It results in the biological and behavioral differences between subjects[6]. Biometric research is about how to measure these characteristics and utilize them in courtesy.

The common characteristics we have explored include fingerprint, iris, face, voiceprint, and gait. Due to the evolution of image sensors, mobile computers, and computer vision algorithms, fingerprint recognition and face recognition technologies have made significant progress in recent years.

Fingerprint

The finger skin is different from the skin on the face or arm. Finger skin has ridges[7] and no hair. The fingerprints are distinct; they may wrinkle with age, but the ridge pattern does not change. Therefore, fingerprints could be the factor for identity verification. Here we only discuss its civilian use. There are tons of fingerprint scanners, designed for attendance machines and personal computers, in our life. The fingerprints found at the crime scene are difficult to extract, and also need different algorithms to handle the low-quality samples.

A fingerprint scanner usually has two parts, sensor and digital signal processor (DSP). Most smartphones use a capacitive sensor. When a fingerprint presses the chip's surface, the sensor will generate a charge difference and save the charge distribution image. The attendance machines use the optical sensor to capture the finger's reflected light and save the image. The optical sensor usually has a higher power. However, when some narrow bezel phones move the sensor under the screen, they have to use the optical sensor. The specific DSP chip can denoise and normalize the fingerprint image. The last step is feature extraction and matching. If the detected features match the ones at the enrollment stage, the user's identity is verified.

Human fingerprints are different from physical keys, as the fingerprints are inherent and cannot be shared. However, people can make lifelike silicone fingers, and this technology has made the fingerprint system vulnerable. For example, the attendance machine is unreliable if there is no surveillance system for reference. Some research labs are exploring liveness detection and anti-spoofing algorithms.

Face

Face appearance is not the most distinct in biometric characteristics. However, face recognition is hand-free and contactless. This technology has emerged since 50 years ago. The eigenface [9] is a significant milestone. It uses the principal components analysis (PCA) method to find the most common facial features among the face image pool. Then we can compute a distinct linear combination of these features for the queried face image, see Figure 7. The k-nearest neighbors (k-NN) algorithm can find the matched face in the database.

The evolution of face recognition technology almost follows the evolution of computer vision. The algorithms, such as local binary pattern (LBP) and sparse representation, have improved face recognition accuracy and robustness. In recent years, the data-driven machine learning approaches, like deep neural networks (DNN), have shown tremendous success in computer vision applications. The deep convolutional neural network (CNN) based models have outperformed the traditional methods in almost every visual computing task. It has also been confirmed that CNNs work very well for face-related tasks including face recognition [11], [12] and facial expression recognition [13]. FaceNet [12] is the least recent milestone, and it uses CNN as the backbone structure.

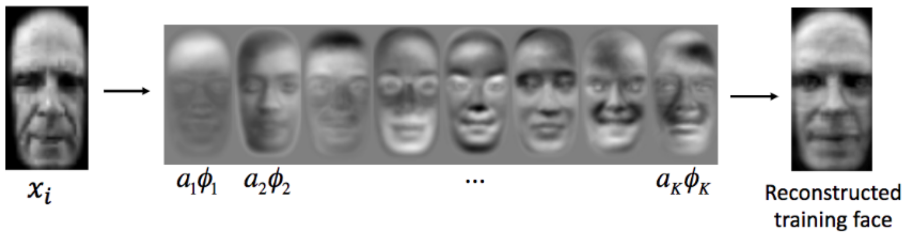


Figure 7: Face representation using Eigenfaces[10].

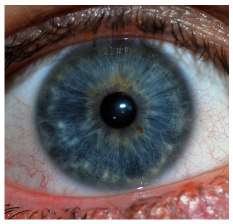


Figure 8: An example image of the human iris[15].

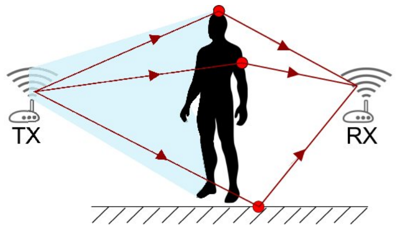


Figure 9: Radio Human Biometrics[17].

Similar to fingerprint recognition technology, researchers are exploring the anti-spoofing algorithms for biometrics. There are also some solutions using non-camera sensors. For example, Some apple models use a dot projector and infrared cameras to generate the 3D human face model. This approach is robust but raises the hardware cost.

Trends in Biometrics

The COVID-19 pandemic has affected this area. As many people wear facial coverings in public zones, face verification becomes disabled. However, iris recognition technology still works. The iris is the circular part between the black pupil and the white sclera. As Figure 8 shows, it contains interlaced spots, filaments, crowns, stripes, crypts, etc. While the human face changes throughout life and identical siblings could have the same face appearance[14], the iris is distinct between identical siblings and does not change. As reading an iris needs a high-quality infrared camera, only some Samsung phones employ this feature. The market situation could change when the pandemic becomes regular.

In the recent release of iOS, when the sensor has detected a mask on the human face, the keypad pops for PIN code input. Some companies [16] have started research on identity recognition with partially-shown faces. Cornell researchers [17] have invented earphones with cameras to observe the contour of the cheeks. We can expand this technology to do identity verification. [18] proposed a novel concept of radio biometrics. We can implement the accurate human identification and verification using commercial Wi-Fi devices in a through-the-wall setting by time-reversal (TR) technique, see Figure 9.

Ethics in Human Identification

In the Keys section, we introduce the identification method using physical devices (keys). The most significant disadvantage is that the keys could be lost or stolen. Some research work has implanted the RFID tag into the human body through surgery [19]. Then, the ownership verification process becomes the biometric verification process. A second-party could track and recognize the human without any effort. Those subjects can exchange or modify their inherent characteristics, resulting in severe damage to the subject. If the surgery is irreversible, that will be worse.

In the Password section, we clarify that knowledge-based identification is secure if we do not save it somewhere or give a third-party a chance to see it. This conclusion becomes incorrect in a brain-computer interface (BCI). The computer could be able to read all knowledge we have for identification with non-intrusive BCI technologies.

CONCLUSION

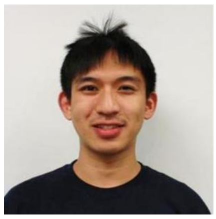
This article has reviewed the advantages and disadvantages of the three kinds of identity verification methods. It introduces the RFID technology in smart IC cards, password manager software, and iris recognition technology. These new approaches are much more reliable than their predecessors and are more diversified. We have seen that some novel approaches, like radio biometrics, are promising and will give us more choices in the future.

ACKNOWLEDGMENT

We thank Dr. Sean Warnick, Michael DeBuse and all other instructors for making this class wonderful.

REFERENCES

1. W. J. Infosino, "Universal magnetic stripe card," Apr. 6 2004. US Patent 6,715,679.
2. T. Ohkawa, M. Yuyama, H. Yoshigi, T. Oonishi, and K. Watanabe, "Rfid (radio frequency identification) and ic card," Dec. 6 2005. US Patent 6,972,662.
3. Maxpayne473, "Rfid," 2016. https://commons.wikimedia.org/wiki/Main_Page.
4. L. Myers, "Your secret question may not be so secret," 2012. https://www.intego.com/mac-security-blog/your-secret-question-may-not-be-so-secret-easy-to-guess-password-retrieval-questions-you-should-avoid-and-why/.
5. G. Ye, Z. Tang, D. Fang, X. Chen, K. Kim, B. Taylor, and Z. Wang, "Cracking android pattern lock in five attempts," in *NDSS*, 2017.
6. A. K. Jain, K. Nandakumar, and A. Ross, "50 years of biometric research: Accomplishments, challenges, and opportunities," *Pattern recognition letters*, vol. 79, pp. 80–105, 2016.
7. M. Trauring, "Automatic comparison of finger-ridge patterns," *Nature*, vol. 197, no. 4871, pp. 938–940, 1963.
8. A. K. Jain, A. A. Ross, and K. Nandakumar, *Introduction to biometrics*. Springer Science & Business Media, 2011.
9. M. A. Turk and A. P. Pentland, "Face recognition using eigenfaces," in *Proceedings. 1991 IEEE computer society conference on computer vision and pattern recognition*, pp. 586–587, IEEE Computer Society, 1991.
10. J. C. Niebles and R. Krishna, "Cs131 computer vision: Foundations and applications," 2019. http://vision.stanford.edu/teaching/cs131_fall1920/slides/13_LDA_fisherfaces.pdf.
11. O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," in *Proceedings of the British Machine Vision Conference (BMVC)*, pp. 41.1–41.12, 2015.
12. F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 815–823, 2015.
13. Z. Yu and C. Zhang, "Image based static facial expression recognition with multiple deep network learning," in *Proceedings of the 2015 ACM on international conference on multimodal interaction*, pp. 435–442, 2015.
14. K. W. Bowyer and P. J. Flynn, "Biometric identification of identical twins: A survey," in *2016 IEEE 8th international conference on biometrics theory, applications and systems (BTAS)*, pp. 1–8, IEEE, 2016.
15. W. contributors, "Iris (anatomy)," 2020. https://en.wikipedia.org/wiki/Iris_(anatomy).
16. L. Song, D. Gong, Z. Li, C. Liu, and W. Liu, "Occlusion robust face recognition based on mask learning with pairwise differential siamese network," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 773–782, 2019.
17. T. Chen, B. Steeper, K. Alsheikh, S. Tao, F. Guimbretière, and C. Zhang, "C-face: Continuously reconstructing facial expressions by deep learning contours of the face with ear-mounted miniature cameras," in *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, pp. 112–125, 2020.
18. Q. Xu, Y. Chen, B. Wang, and K. R. Liu, "Radio biometrics: Human recognition through a wall," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 5, pp. 1141–1155, 2017.
19. K. R. Foster and J. Jaeger, "Ethical implications of implantable radiofrequency identification (rfid) tags in humans," *The American Journal of Bioethics*, vol. 8, no. 8, pp. 44–48, 2008.



Z. Casey Sun is a PhD student in Electrical and Computer Engineering at Brigham Young University.

Natural Language and Machines: the Evolution of Natural Language Processing

C. L. Byun
Brigham Young University

***Abstract*—Humans have long shown interest in having machines use and produce human-like language. This idea, known as natural language processing (NLP), has been a part of computer science for 70 years. The purpose of this paper is to explore the origins and evolution of machine interaction with and use of natural human language. The importance of this idea was seen from the early days of computer science and its evolution is also tightly intertwined with areas of linguistics and artificial intelligence (AI). We trace the watershed events and principal players through each of the three major eras in the history of NLP in an attempt to clearly illustrate how and why machine interaction with natural language has evolved throughout the decades. We also explore how innovations in other fields, particularly within the fields of AI and linguistics, have led to innovations within the field of NLP.**

■ THE IDEA of having a machine use and or produce natural language was conceived well before the 20th century. The invention of mass-producing movable type and the printing press by Johannes Gutenberg in the 1400s could perhaps be considered the first fledgling step of NLP. Another example of early interest in machine usage of language is the writer automata, built by Jaquet-Droz sometime in the mid-1700s, an intricate machine which can be be programmed to write any text with 40 or fewer characters [1]. The concept of Machine Translation (MT), a major area of NLP, in which a machine is used to translate speech or text from one language to another, was one of the earliest areas to be explored in computer science. MT appears to have made its debut in fiction as early as 1945 in Murray Leinster's scifi novelette, "First Contact," in which a universal translating device is used to help different species communicate with one another [2].

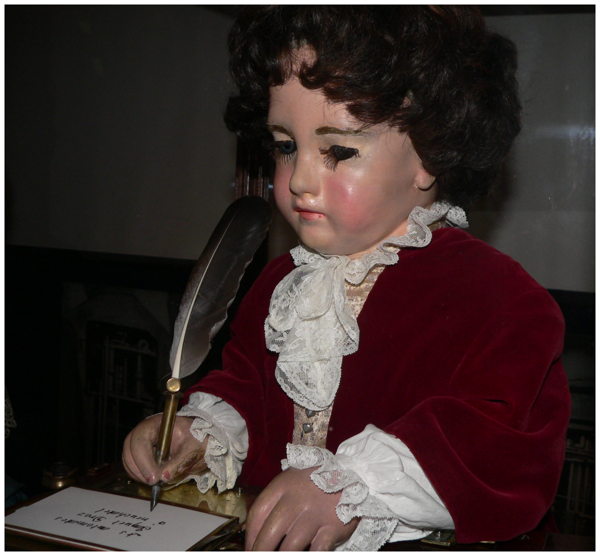


Figure 1. Writer automaton built by Jaquet-Droz. (From [3]).

Today machines can use and produce language in a myriad of applications, though, this language is still far from what could be considered truly natural language. AI systems can be trained to create text that reads in the same style as a real person [4]. Topic models are NLP systems that can be used to extract topics from collections of texts [5]. NLP can also be used to read the current mood of Twitter [6]. The result can then be used to change the color on a mood lamp to reflect that mood [7]. Google translate can be used to automatically translate a restaurant menu written in Chinese into English or to help two people with no shared language to communicate [8]. None of these systems are yet perfect, but the progress that has been made since the first conception of NLP is astounding. What happened in between the birth of the idea and its current vibrant applications? How did we get from scifi dreams of automatic and universal machine translation to, at least in some respects, near-reality? The purpose of this paper is to answer these questions.

There are three main periods in the history of NLP: the Symbolic, Statistical, and Neural eras [9], with each era named after the types of NLP systems that were typically built during that time period. We will explore each of these eras, highlighting the people and innovations that influenced them and led to the boom the field is experiencing today. The areas of linguistics, computer science, and AI have all either been influenced by NLP or have been influential on NLP. As such, we will also explore the evolution of each of these areas in tandem, insofar as it is beneficial to paint a clearer picture of the evolution of NLP.

The Symbolic Era: 1950s - 1990s

The Turing Test

Alan Turing, often called the father of computer science, was one of the first to draw attention to the important role NLP would play in a true AI system [10]. His seminal paper, ”Computing Machinery and Intelligence,” published in 1950, laid out the idea of the Turing test as a way to test the intelligence of an AI system. The Turing test consists of three key players: two humans and the system to be tested. One of the humans acts as a mediator. Each of the three players are to be kept in separate locations and interact with one another only through a computer, by typing in questions and responses. The mediator’s role is to ask questions of both the other human and the AI system and use their responses to make a guess as to which is the human and which is the AI system. If the AI system is able to consistently trick the mediator into not knowing which is which, it can be said to have passed the test. In order to pass the test such a system must be able to demonstrate both automated creation and comprehension of natural language. The desire to create a system that can truly be said to pass the Turing test has been a motivator for continued work within NLP and AI and inspired some of the systems which will be discussed later in this paper.

Machine Translation and a Slow Start

The first notable foray into NLP came just four years after Alan Turing’s Turing test proposal. In 1954 the Georgetown-IBM experiment, headed by Cuthbert Hurd and Leon Dostert, produced the first MT system [11]. Their system took Russian sentences as input. They were input to the system by a user with absolutely no Russian fluency. The system then translated those sentences into English and printed the translations as output. Early MT systems like this one generally worked by having a bilingual dictionary used to map words from the first language to the second. Then, the system would have a second step that would rearrange the order of the translated words to fit the word order of the target language. While these early systems were rudimentary (the Georgetown-IBM system could only provide accurate translations for around sixty carefully chosen sentences) it gave insight into the potential power of using machines to solve natural language problems, and further research was begun.

In the summer of 1956 John McCarthy, along with Marvin Minsky, Claude Shannon and several other important early figures in computer sicence, held the Dartmouth Conference [12]. The conference had 11 attendees all coming together to discuss ideas about intelligent machines. This event is considered to be the birth of AI as a field. Also of note is that NLP was mentioned in the proposal as one topic of study the group would be interested in exploring.

Noam Chomsky was another notable figure from this era of NLP. His 1957 *Syntactic Structures* helped NLP researchers realize the need for mainstream linguistic knowledge within NLP [13] [14]. This work introduced the idea of 'universal grammar,' the idea that certain aspects of language structure are built into the human brain and are universal across all languages and people. Chomsky's later linguistic theory on generative grammar, also known as transformational grammar, also had a strong influence on the field [15]. This theory purported that natural languages have two structures: deep structures and surface structures, which connect phonetic rules with sounds and connect words with meaning, respectively. The connection between these two structures is generative grammar. The idea of generative grammar models language as a system of hard rules. Generative grammars can be used to represent syntactic structures as abstract symbols. A series of simple rules can then be applied to a generic input symbol to generate a desired output. The theory, however, struggled to handle ambiguities inherent in natural human language and began to fall out of favor within linguistic and NLP communities in the 1980s.

It was initially believed that MT would be an easy problem to solve, with one of the participants in the Georgetown-IBM experiment claiming in 1954 that this could likely be accomplished within three to five years [11]. However, it has now been seventy years since the dawn of NLP and neither MT, nor NLP in general are considered solved problems.

Further research was conducted in MT following the success of the Georgetown-IBM experiment, but progress was slow. The systems created were based on hand-calculated and hard-coded rules. These systems were tedious to update and they quickly became cumbersome to work with if more than a few rules were used. Because of this they were extremely difficult to maintain.

Then, the Automatic Language Processing Advisory Committee (ALPAC) report of 1966 served a devastating blow to MT [16]. The report was intended to provide the United States government with an update on the status of NLP and MT research. It portrayed the progress in MT in an unfavorable light and encouraged more emphasis on foundational research in the basics of handling language data and in MT research. For example, one proposal in the report was for research to speed up human translations of texts, rather than focusing on purely machine-based translation systems. As a result of the ALPAC report, funding for MT was severely cut. This area of NLP received little attention for the next two decades.

Progress in Other Areas

While MT was suffering from budget cuts, other areas of NLP did see progress. Similar to the early MT systems, systems in other areas of NLP also relied on hard-coded, hand-calculated rules during this era. While this method was tedious and difficult to maintain, some of the systems created during this era were still able to perform impressive tasks. In some cases they were even able to convince users that they had some level of intelligence. Perhaps the best example of this was Joseph Weizenbaum's chatbot, ELIZA [17] [18].

A chatbot is a computer program designed to interact with human users. Typically a human will enter a question or statement through an online interface and the chatbot will respond. The earliest chatbots took text as input. ELIZA was one of the earliest notable chatbots.

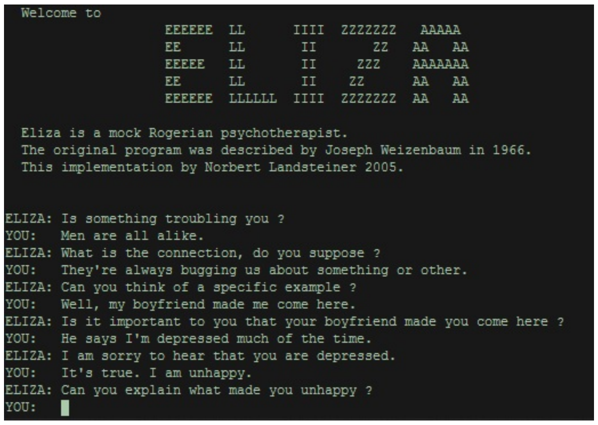
Weizenbaum's intention in creating ELIZA was to show that communication between humans and computers is superficial. He named the program after the character Eliza Doolittle in George Bernard Shaw's *Pygmalion* [19]. The character Eliza comes from a lower-class background, which her natural language clearly advertises. Throughout the play she learns to blend in with the aristocracy, but still sometimes struggles to understand them.

Similarly, the system ELIZA was designed to give an illusion of being human, though it had no real way of actually understanding humans. A user would enter statements on a keyboard as input and ELIZA would respond to the user. Weizenbaum wrote programs for several different scenarios, but the one ELIZA is best known for was one in which the system played the role of a Rogerian psychotherapist and the user plays the role of a patient. This scenario was intended to play out as the first meeting between the therapist and patient. Weizenbaum chose this scenario because he thought it would be fairly easy to mimic a Rogerian psychotherapist's style, as they tend to follow a pattern of mirroring back their patients' statements with different phrasing. He also thought this scenario would be easy because, in this setting if a patient is telling their therapist about how they spent their weekend on a boat and the therapist asks the patient to tell them about boats, instead of assuming the therapist knows nothing about boats, the patient will likely assume the therapist has some deeper intention behind their question. Weizenbaum hypothesized this assumption that the patient would help maintain the illusion that ELIZA was human-like [18].

ELIZA had no understanding of context, but relied on a system of rules to choose its responses. It would pick out keywords from the user's input. Those keywords could then be used to fill in a response pattern to give the illusion of contextual understanding. If no keywords were found in the input, ELIZA still had several generic responses, like ”I see,” that could be used to continue the illusion. Weizenbaum had expected his illusion to break down fairly quickly, however, it proved more effective than he had anticipated. Despite ELIZA's having no contextual grasp and no human emotion, some users who interacted with the system reported feeling that ELIZA could understand them, as if they felt an emotional connection with the program. Of this phenomenon Weizenbaum wrote ”I had not realized ... that extremely short exposures to a relatively simple computer program could induce powerful delusional thinking in quite normal people” [20].

While ELIZA likely would not have passed the Turing test, the fact that it was able to illicit an emotional response in multiple human users shows remarkable progress in machine use of natural language. ELIZA's output was generally grammatically correct and coherent. The system was also able to respond to novel input, rather than a limited number of carefully curated selections. Ultimately, ELIZA is notable because it was able to fool some human users into thinking it was intelligent.

A similar system from this time period was PARRY. PARRY was another chatbot, designed by Kenneth Colby in the early 1970s [22]. Colby was a psychiatrist interested in studying how computer science could benefit psychiatry, particularly by helping broaden understanding of mental illness. PARRY was inspired by ELIZA, but instead of being designed to act as the psychiatrist, PARRY was designed to play the part of a patient with paranoid schizophrenia.



```
 Welcome to
                   EEEEEE  LL      IIII  ZZZZZZZ   AAAAA
                   EE      LL       II       ZZ   AA   AA
                   EEEEE   LL       II     ZZZ    AAAAAAA
                   EE      LL       II    ZZ      AA   AA
                   EEEEEE  LLLLLL  IIII  ZZZZZZZ  AA   AA

 Eliza is a mock Rogerian psychotherapist.
 The original program was described by Joseph Weizenbaum in 1966.
 This implementation by Norbert Landsteiner 2005.


ELIZA: Is something troubling you ?
YOU:   Men are all alike.
ELIZA: What is the connection, do you suppose ?
YOU:   They're always bugging us about something or other.
ELIZA: Can you think of a specific example ?
YOU:   Well, my boyfriend made me come here.
ELIZA: Is it important to you that your boyfriend made you come here ?
YOU:   He says I'm depressed much of the time.
ELIZA: I am sorry to hear that you are depressed.
YOU:   It's true. I am unhappy.
ELIZA: Can you explain what made you unhappy ?
YOU:   █
```

Figure 2. Sample conversation with ELIZA from a later implementation produced by Norbert Landsteiner. (From [21]).

PARRY was actually tested on a version of the Turing test and was able to fool half of the mediators, who were trained psychiatrists, into believing the system was human. While this success could be the result of several other factors besides PARRY truly demonstrating intelligence, the fact that the system was able to consistently fool half of the mediators was a significant step forward.

One amusing side note - ELIZA and PARRY were allowed to 'meet' and have 'conversations' with one another on several occasions, with ELIZA acting as the psychiatrist and PARRY as the patient [22].

The Ball Gets Rolling

Research in NLP began to pick up in the 1980s and into the early 1990s. In the early 1980s there was still heavy emphasis on using detailed, hand-written rules for NLP systems. We will look briefly at two systems from this time period: Rollo Carpenter's Jabberwacky and William Chamberlain and Thomas Etter's Racter [23] [24].

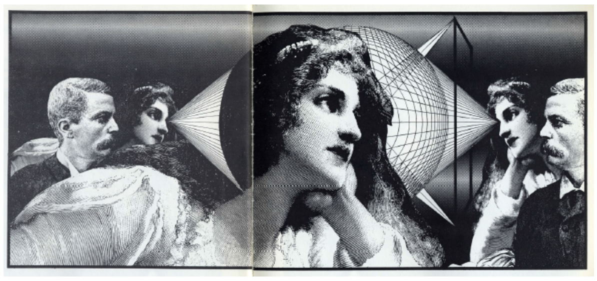


Figure 3. "Helene spies herself in the enthralling conic-section yet she is but an enrapturing reflection of Bill. His consciousness contains a mirror, a sphere in which to unfortunately see Helene. She adorns her soul with desire while he watches her and widens his thinking about enthralling love. Such are their reflections." (From [24]).

Similar to ELIZA and PARRY, Jabberwacky was a chatbot designed to carry on conversations with users through use of the keyboard. Carpenter's sole intention in creating Jabberwacky was to design a system to pass the Turing test. In 2011 one evolution of this chatbot was entered into a formal Turing test with the human mediators who judged that iteration of the system to be 59.3% human, while the human participants were only judged to be 63.3% human [25]. That version of the Jabberwacky chatbot was named Cleverbot and is still around today as an openly-available online chatbot [23].

Racter, on the other hand, was notably used to generate an entire book, *The Policeman's Beard Is Half Constructed*, in 1984. The code used to generate the book was never released and there is speculation as to the sophistication of the program. Still, an entire book generated by a computer in the 1980s was an impressive feat and calls to mind more recent efforts, like the 2019 project Booksby.ai [26]. Booksby.ai is essentially a bookstore offering books entirely created by AI, from the cover art to the reviews. The language from both systems falls very short of natural language, but they do show significant steps forward. See **Figure 3** for an excerpt from Racter's book.

Shifts within the field and innovations in other areas, like corpus linguistics and computer engineering, led to a move away from the symbolic approaches and towards statistical systems. Increases in computer power made machine learning algorithms more viable, while increases in

Figure 4. From left to right: Alan Turing, Noam Chomsky, and Joseph Weizenbaum. (From [27] [28] [29]).

computer storage capacities made it easier to store large collections of linguistic data needed for statistical approaches. Additionally, steps forward within the field of corpus linguistics made it easier to acquire large collections of language data. Finally, within the field of NLP itself, there was a general push towards more general approaches to solving NLP problems. These innovations, working in tandem, eventually led to the second era of NLP: the statistical era.

The Statistical Era: 1990s - 2010s

A Brief Introduction to Corpus Linguistics

Considering that corpora, or large collections of real world language data, began to play a key role in NLP during the statistical era, we will now make a brief inspection of the history of corpus linguistics. The word *corpus* comes from Latin and means 'body.' In corpus linguistics it is used to refer to a body, or collection, of texts. One main focus of corpus linguistics is to compile collections of text samples that accurately reflect how frequently each word appears and how words pattern within a chosen language. This is because corpora (the plural of *corpus*) are used in corpus linguistics to study languages and how they work. If a corpus is not truly representative of the language it is intended to represent, then it cannot be used to gain an accurate picture of word usage in that language.

In the past corpora have typically been collected for a specific purpose. A corpus may be a collection of texts from a specific genre, time period, geographic area, or any other grouping. Mark Davies' Corpus of Historical American English (COHA) is an example of a corpus designed to be used for studying diachronic changes in American English [30]. COHA contains text from many sources ranging from the 1810s all the way up through the 2000s.

Before the advent of computers, corpora had to be compiled by hand. Modern corpora that still predated computers were often collections of odd or interesting language usages an individual scribbled down after coming across it in conversation or while reading. One early corpus linguist, Otto Jespersen, famously kept his text samples on little strips of paper in shoe boxes strewn throughout his villa [31]. It is easy to imagine that collecting, storing, and analyzing these early corpora was tedious. It would also be extremely difficult to store a very large corpus in this format, let alone use it. The creation of computers changed this.

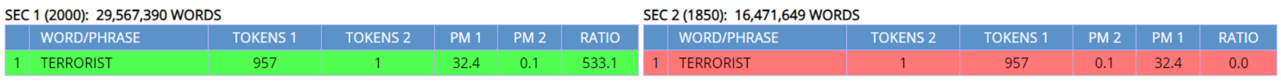
As technology has advanced, computers have made it possible to store very large corpora using relatively little space. Computers have also made it possible to quickly and more accurately analyze information within corpora, in addition to making it easier and faster to collect corpora - as typing is certainly faster and easier than writing language samples out by hand.

One of the earliest and most well-known electronic corpora is the Brown Corpus [32] [33]. The Brown Corpus was compiled in the 1960s at Brown University by Henry Kucera and W. Nelson Francis. It is a collection of American English text samples taken from various genres. Every sample in the corpus is from the year 1961. The corpus consists of approximately one million words.

Electronic corpora have come a long way since the Brown Corpus. Today the Corpus of Contemporary American English (COCA), another of Mark Davies' online corpora, consists of more than one billion words, sampled from eight different genres [34]. COCA is periodically updated by Davies, so that it is continually growing and can be relied on to contain truly contemporary samples of American English.

Before we return to our study of the evolution of NLP it is worth noting the limitations of corpora. Natural language is messy. It was perhaps a lack of fully understanding this that led early researchers in MT to believe it could be solved within a few short years. The truth is human languages are constantly evolving, pattern differently in different scenarios, are used differently by different groups of people, and often have exceptions that break strict rules we try to tie them down with. As previously noted, historically, corpora have been designed with specific purposes in mind. The text samples collected are typically taken from a pre-determined form of language, like 'spoken' or 'science fiction.' The information about what types of language a corpus represents is important to know when using the corpus. For example, if you were interested in studying how the word *terrorist* was used in American English in 1850 versus 2000, you would want a corpus that had text from American English and was sampled from the desired years. It would also be helpful if the corpora were annotated with the years the texts were sampled from so you could easily make your desired comparison. To use Mark Davies' corpora as an example, COHA would work well for this use case because it is annotated for diachronic analysis and it does contain American English. COCA, on the other hand, would not be a good corpus to use for this use case because it only has samples going back to 1990. COCA would, however, be an excellent choice if you were interested in looking at whether the word *terrorist* appears in contemporary American English more often in samples of spoken English versus samples taken from news networks. **Figure 5** shows the results of the COHA query. Unsurprisingly there are far more occurrences in the year 2000 (957) than there are in the year 1850 (1). **Figure 6** shows the results of the COCA query. In COCA we can compare the results across different registers (areas of language use) and time periods. We can clearly see there are far more instances of the word in spoken English (7367) than in the news network category (4503). Looking at the left of **Figure 6** we can also clearly see a significant spike in usage during the period from the year 2000 through the year 2004.

Corpora are becoming increasingly large. With the advent of the internet and the rise of social media, enormous amounts of textual data are freely available to anyone willing to put in the effort to collect it. It should be remembered, however, that the corpora created by these vast collections of data may not be representative of natural language in general. The majority of natural language data available online is written.



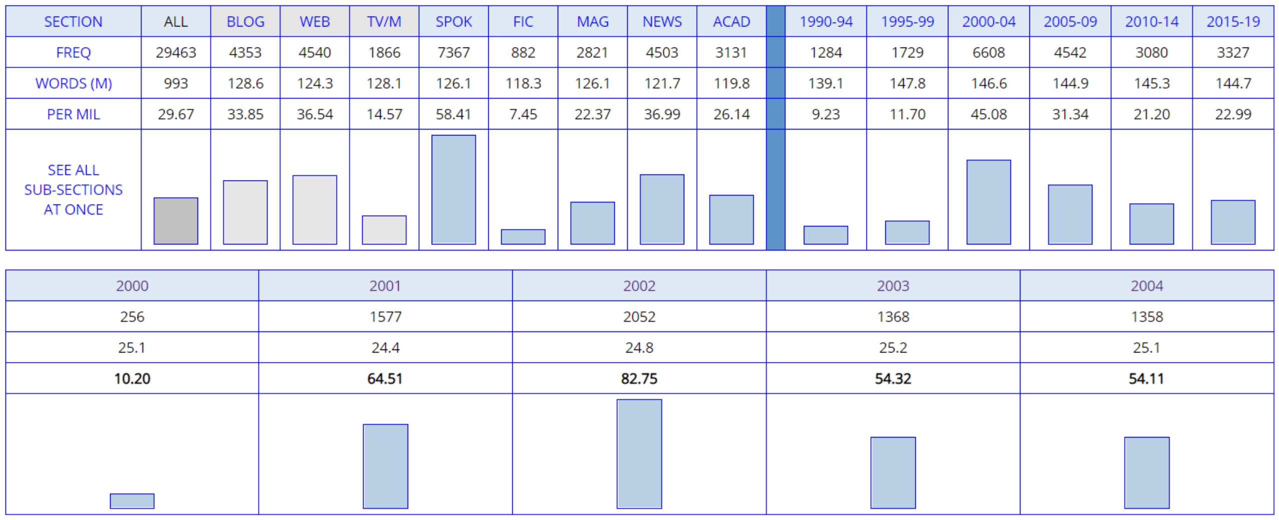
SEC 1 (2000): 29,567,390 WORDS

	WORD/PHRASE	TOKENS 1	TOKENS 2	PM 1	PM 2	RATIO
1	TERRORIST	957	1	32.4	0.1	533.1

SEC 2 (1850): 16,471,649 WORDS

	WORD/PHRASE	TOKENS 2	TOKENS 1	PM 2	PM 1	RATIO
1	TERRORIST	1	957	0.1	32.4	0.0

Figure 5. Result of querying *terrorist* for 2000 and 1850 in COHA. (From [30]).



SECTION	ALL	BLOG	WEB	TV/M	SPOK	FIC	MAG	NEWS	ACAD		1990-94	1995-99	2000-04	2005-09	2010-14	2015-19
FREQ	29463	4353	4540	1866	7367	882	2821	4503	3131		1284	1729	6608	4542	3080	3327
WORDS (M)	993	128.6	124.3	128.1	126.1	118.3	126.1	121.7	119.8		139.1	147.8	146.6	144.9	145.3	144.7
PER MIL	29.67	33.85	36.54	14.57	58.41	7.45	22.37	36.99	26.14		9.23	11.70	45.08	31.34	21.20	22.99
SEE ALL SUB-SECTIONS AT ONCE																

2000	2001	2002	2003	2004
256	1577	2052	1368	1358
25.1	24.4	24.8	25.2	25.1
10.20	**64.51**	**82.75**	**54.32**	**54.11**

Figure 6. Result of querying *terrorist* in COCA. (From [34]).

Thus, while corpora created from online data is absolutely useful and beneficial, it may not be representative of, say, spoken language, and therefore may not be useful if you are wanting to use a corpus to model spoken language. These giant corpora, however, have proven to be extremely useful in the neural era of NLP, which we will discuss shortly.

Winter's End and the Rise of Machine Learning

In the field of AI, the period extending from 1974 to 1980 is known as the first AI winter [35]. Similar to the early days of MT, there was initially great optimism and a flurry of research in the field of AI. When the various branches of AI research failed to deliver the desired results, however, research funding was cut. This led to a six-year dark age when little progress was made.

The first AI winter finally began to thaw in 1980 and continued to do so throughout the early 1980s as innovations led to renewed interest in neural networks and machine learning. Chomsky's theories also began to fall out of favor during this period. He had been skeptical of using statistical models, but as his theories became less popular, more researchers turned to exploring statistical methods. During this time there was also an increased push for greater emphasis on the quantitative evaluation of systems. NLP systems during this period generally relied heavily on the use of corpora and this era also saw NLP become increasingly intertwined with machine learning [36]. The shift away from using intricate, tediously hand-calculated rules towards using statistical inference allowed for systems that could 'learn' the rules of language on their own. This allowed for more general solutions to NLP problems than were previously possible.

MT saw renewed interest and success during this period. Researchers were able to take advantage of multilingual corpora to train systems. One notable MT system from this period is Babel Fish, the first web-based translation tool [37]. It was named after the universal translation 'device' from Douglas Adams' "The Hitchhiker's Guide to the Galaxy," a multi-media series depicting the intergalactic adventures of a human named Arthur Dent [38]. In the series the babel fish was described as being "small, yellow and leech-like," and feeding on brainwaves. The characters would insert these fish into their ears and could instantly be able to perfectly understand anything said in any language. This description appears to

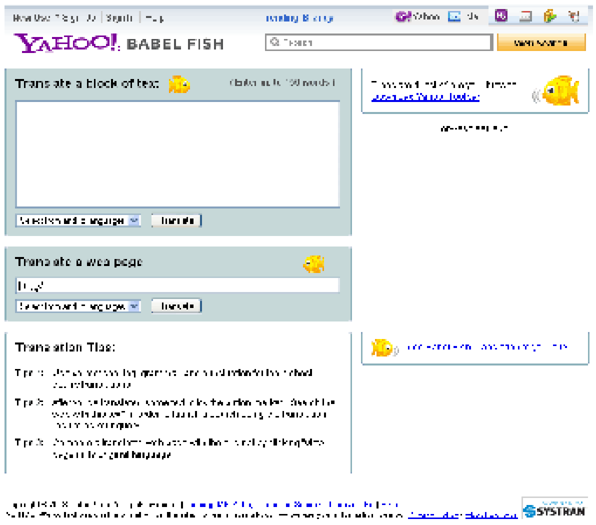


Figure 7. Original Babel Fish interface. Note the ”small, yellow” fish prominently displayed throughout the interface as a nod to Adams’ series. (From [39]).

have informed the design of Yahoo!’s Babel Fish website, as seen in **Figure 7**. The system was created by Yahoo! in 1997 and could translate text between thirteen different languages. While still a far cry from its namesake’s ability to automatically translate from any language to any other, it was a huge step forward from the Georgetown-IBM experiment. It could be considered a precursor to today’s Google Translate.

DragonDictate (later known as NaturallySpeaking) was another notable system from this time period [40]. James Baker and his wife, Janet M. Baker, founded the company Dragon Systems in the early 1980s and worked to develop systems for speech recognition. DragonDictate was the first iteration of the Bakers’ system. One of the major shortcomings of this system (and early speech systems in general) was its inability to distinguish between when one spoken word ended and another began, thus requiring users to speak input slowly, one word at a time. This shortcoming was later resolved and the next iteration of the system, NaturallySpeaking, was released in 1997 as the first speech recognition system able to accept continuous speech as input.

Supervision

Many statistical methods focused on using electronic corpora. One of the benefits of using traditional corpora is that they are often annotated in such a way that they can be useful for the task they were designed to be used for. As this era evolved, however, the internet became increasingly prominent in daily life. People began to use the internet for almost everything - buying, selling, rating products, and communicating, in addition to many others. As such, vast amounts of raw text data became available. The raw data of the internet, however, often has little or no accompanying annotation. This made it difficult for use in earlier ’supervised’ approaches that required input data to have annotations. This motivated a stronger focus on approaches that did not require annotations, what are known as ’unsupervised’ approaches.

One example of these unsupervised approaches is David Blei’s Latent Dirichlet Allocation (LDA) topic model, introduced in 2003 [41]. Topic models are statistical models used to determine what abstract topics documents belong to. LDA is based on two assumptions: that each document belongs to multiple topics and that similar documents (documents belonging to the same topics) will use similar words. The topics chosen by the model are abstract, meaning the model does not assign an actual name or definition to the topics. Instead, it represents each topic as a distribution of the words appearing in it and leaves it up to the user to interpret the assigned topics. It should also be noted that LDA requires the user to determine the number of topics the model will find in advance.

LDA works by first randomly assigning each word in each document a topic. Then, it chooses a word from a document and updates the topic assignment for that word. It does so by going through each document and, for each word in that document, going through each topic and calculating two things. First, it calculates the probability of seeing the current topic if we are in the current document and second, the probability of seeing the current word based on the current topic. The results are then multiplied to get the probability that the current word belongs to the current topic. Finally, the word is assigned the topic it has the highest probability of belonging to. This process is repeated many times until the topic assignments are relatively stable.

LDA has become a staple topic modeling algorithm and is still widely used today.

The Neural Era: 2010s - Present

The 2010s brought incredible steps forward in NLP as neural networks began to be utilized to solve various natural language problems. Neural networks are intended to emulate the human brain. One of their earliest and most rudimentary forms, the perceptron, is an algorithm invented in 1958 by Frank Rosenblatt [42].

Figure 8 gives a graphical representation of a perceptron. The input values (on the left side of the image) are multiplied by weights. The weights help determine which values are of higher or lower importance. That is, a value with higher importance would be multiplied with a higher weight and vice versa. The results are combined into the net input function, which is typically the sum of all the weighted inputs. This is then passed into the activation function, which will typically check to see if the weighted sum from the net input function is above a certain threshold. If it is, then the perceptron ’fires’, similar to a neuron in the brain. Typically ’firing’ would equate to passing a 1 to the output and ’not firing’ would equate to passing a 0 to the output. The output of the perceptron is then checked against the desired output and, if the actual output and the desired output do not match, the weights of the perceptron are adjusted to hopefully give a more accurate result for its next input. The standard equation for updating the weights is calculated by subtracting the actual output from the desired output and multiplying the result by a learning rate, usually a decimal value that determines how quickly the weights are changed, and by the input value. This is calculated for each weight. The results are added to the current weight values to find the new weights.

In a basic neural network there will typically be several layers comprised of many of these ’neurons’. In a ’fully-connected’ neural network, all of the neurons from one layer will have weights connecting each one to all of the neurons in the next and previous layers. Deep neural networks have many layers.

There have been many innovations in the area of deep neural networks over the past decade that have led to incredible steps forward in NLP. Two of the notable innovations in NLP with neural networks have been the introduction of word embeddings and transformers.

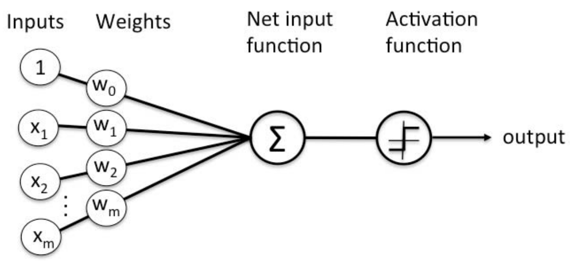


Figure 8. Basic model of a perceptron. (From [43]).

Word embeddings are a way of representing words as lists of numbers, also known as vectors. Their popularity increased sigificantly in 2013 when Word2Vec was introduced by a team from Google [44] [45]. Their approach uses a neural network to model word associations as vectors and is much faster than earlier approaches. One observation that led to increased interest is that Word2Vec was shown to be able to detect synonymous and analogous words by looking at the vector representations. One famous example is that the vector for the word *queen* can be found by subtracting the vector for *man* from the vector for *king* and adding the result to the vector for *woman*. Word2Vec is still widely used in NLP today.

Transformers were first introduced in 2017 [46]. They use what is called an ”encoder-decoder” design. This means they have two main parts: an encoder and a decoder. The encoder itself also has two parts. First it has what is called a ”self-attention” layer. This layer looks both at the specific word it is encoding and all the other words in the input sample. The second part of the encoder layer is a neural network. The output from the self-attention layer is fed as input into the neural network layer and the result is then fed into the decoder layer. The decoder layer has the same exact setup as the encoder layer, with the exception that it also has what is called an an ”attention” layer tucked between the self-attention and neural network layers. The attention layer helps the model pay attention to important parts in the input sample. Transformers have been popular within NLP because they are able to handle sequential data well, which is something past models have struggled to do. Language is sequential, so having a model that can handle sequential data is useful.

Some of the most notable systems of this era thus far have been Google Translate, Siri, and GPT-3 [47] [48] [49]. Google Translate was initially launched in 2006, but has undergone numerous evolutions since then. It began using a statistical model for MT, but in 2016 this was switched to a neural network system [50]. While not perfect, it can translate input between more than 100 different languages, a vast improvement from the days of Babel Fish. Its services have also been expanded to include spoken and written translations, in addition to translations of text in images captured with a phone camera. It can even translate hand-written text.

Siri is Apple Inc.'s virtual assistant. Siri was initially released in 2011 and has also undergone numerous changes. It can be used to do a variety of tasks on various Apple devices by voice command. Siri can be used to make calls, order products, set reminders, and many other tasks. Over the past decade, Siri has become a household name and a staple of Apple Inc. devices.

GPT-3 is the latest iteration of OpenAI's pre-trained language models. It was first described earlier this year and, similar to its predecessors, is making waves within the NLP community. GPT-3 is a pre-trained language model that can be used to generate very realistic text. Research on the original model, generative pre-training (GPT), was released by Alec Radford et al. in 2018 [51]. This was followed in 2019 by GPT-2 [52]. GPT-2 made use of a transformer model. Results from the model were so impressive that the actual model itself was withheld from release to prevent its being put to nefarious purposes [53]. GPT-3 was introduced in 2020 [49]. The model has yet to be released as of the writing of this paper, but the results have been impressive. It is important to note, however, that these models still have no grasp of context.

CONCLUSION

In summary, we have explored the evolution of how machines interact with and generate natural language. While, at seventy years old, the field is still relatively new, it has come a long way from its earliest beginnings. We began with the early days when a machine could only be used to parrot hard-coded natural language. Then, in the mid-twentieth century, we saw how early programmers used complex, hand-coded rules to begin solving slightly more generic problems, though they were often limited to only being accurate on a small subset of examples. As time went on statistical methods were adopted that allowed language models to begin to infer the rules of language for themselves. Finally, we saw the rise of neural networks, along with the rise of electronic corpora and increased processing power, that have allowed for the more powerful models of today. While we are still far from having a machine that can generate and interpret natural language on a truly human level, in seventy years we have come a long way.

REFERENCES

1. "Mechanical marvels: Clockwork dreams," 2013.
2. M. Leinster, *First Contact*. Analog Science Fiction and Fact, 1945.
3. Wikipedia and the free encyclopedia, online; accessed December 2 and 2020; By Rama - Own work and CC BY-SA 2.0 fr and https://rb.gy/0kqskz. [Online]. Available: https://rb.gy/g9z5rg
4. Botnik, online; accessed December 2 and 2020. [Online]. Available: https://botnik.org/content/harry-potter.html
5. D. M. Blei, A. Ng, and M. Jordan, "Latent dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.
6. Hasan and M. et al., "Emotex: Detecting emotions in twitter messages," 2014.
7. RandomMatrix, "Twitter mood light - the world's mood in a box," online; accessed December 2 and 2020. [Online]. Available: https://www.instructables.com/Twitter-Mood-Light-The-Worlds-Mood-in-a-Box/
8. Whitney and Lance, "Don't speak the language? how to use google translate," *PC*.
9. Wikipedia and the free encyclopedia, online; accessed December 2 and 2020. [Online]. Available: https://en.wikipedia.org/wiki/Natural_language_processing
10. Turing and A. M, "Computing machinery and intelligence," *Mind*, pp. 433–460, 1950.
11. IBM, "701 translator," 1954, online; accessed December 2 and 2020. [Online]. Available: https://www.ibm.com/ibm/history/exhibits/701/701_translator.html

12. McCarthy, J, Minksy, M. L, Rochester, N, Shannon, and C. E, “A proposal for the dartmouth summer research project on artificial intelligence,” 1955, online; accessed December 2 and 2020. [Online]. Available: http://raysolomonoff.com/dartmouth/boxa/dart564props.pdf
13. N. Chomsky, *Syntactic Structures*, 1957.
14. E. D. Liddy, “Natural language processing,” in *Encyclopedia of Library and Information Science and 2nd Ed.* Marcel Decker and Inc., 2001.
15. N. Chomsky, *Aspects of the Theory of Syntax*, 1965.
16. A. L. P. A. Committee, “Language and machines: Computers in translation and linguistics,” National Academy of Sciences, Tech. Rep., 1966.
17. J. Weizenbaum, *Computer Power and Human Reason: From Judgement to Calculation*. San Francisco: W.H. Freeman, 1976.
18. ——, “Computational linguistics: Contextual understanding by computers,” *Communications of the ACM*, vol. 10, no. 8, pp. 474–480, 1967.
19. G. B. Shaw, *Pygmalion*, 1912.
20. D. Hill, *Not So Fast: Thinking Twice about Technology*, 2013.
21. Wikipedia and the free encyclopedia, online; accessed December 3 and 2020; By Unknown author - https://rb.gy/ea25gk and Public Domain and https://rb.gy/psowg9. [Online]. Available: https://en.wikipedia.org/wiki/ELIZA
22. K. M. Colby, “Modeling a paranoid mind,” *The Behavioral and Brain Sciences*, vol. 4, pp. 515–560, 1981.
23. “Cleverbot,” online; accessed December 2 and 2020. [Online]. Available: https://www.cleverbot.com/
24. Racter, *the Policeman’s Beard is Half Constructed*, 1984.
25. “Cleverbot - turing test at techniche 2011,” online; accessed December 2 and 2020. [Online]. Available: https://www.cleverbot.com/human
26. Online; accessed December 5 and 2020. [Online]. Available: https://booksby.ai/
27. Wikipedia and the free encyclopedia, online; accessed December 5 and 2020; By Unknown author and http://www.turingarchive.org/viewer/?id=521amp;title=4 and Public Domain and https://rb.gy/grtkbf. [Online]. Available: https://rb.gy/nzbmor
28. ——, online; accessed December 5 and 2020; By and retouched by Wugapodes and File:Noam_Chomsky_portrait_2017.jpg and CC BY-SA 4.0 and https://rb.gy/a28qgc. [Online]. Available: https://rb.gy/80doze
29. ——, online; accessed December 5 and 2020; By Ulrich Hansen and Germany (Journalist) and Own work and CC BY-SA 3.0 and https://rb.gy/g1cfx4. [Online]. Available: https://rb.gy/ablslk
30. M. Davies, online; accessed December 5 and 2020. [Online]. Available: https://www.english-corpora.org/coha/
31. H. Lindquist, *Corpus Linguistics and the Description of English*. Edinbrgh University Press, 2009.
32. W. N. Francis and H. Kucera, “Brown university standard corpus of present-day american english,” 1979, online; accessed December 5 and 2020. [Online]. Available: http://korpus.uib.no/icame/brown/bcm.html
33. ——, “Computational analysis of present-day american english,” *International Journal of American Linguistics*, vol. 35, 1967.
34. M. Davies, online; accessed December 5 and 2020. [Online]. Available: https://www.english-corpora.org/coca/
35. M. Lim, “History of ai winters,” 2018, online; accessed December 5 and 2020. [Online]. Available: https://www.actuaries.digital/2018/09/05/history-of-ai-winters/
36. P. M. Nadkarni, L. Ohno-Machado, and W. W. Chapman, “Natural language processing: an introduction,” *Journal of the American Medical Informatics Association*, 2011.
37. Wikipedia and the free encyclopedia, online; accessed December 5 and 2020. [Online]. Available: https://en.wikipedia.org/wiki/Babel_Fish_(website)
38. D. Adams, “The hitchhiker’s guide to the galaxy,” 1978-1980.
39. Wikipedia and the free encyclopedia, online; accessed December 5 and 2020 and By Source and Fair use and https://rb.gy/63rvt7. [Online]. Available: https://rb.gy/tlv0n1
40. . F. M. Software, online; accessed December 5 and 2020. [Online]. Available: http://www.dragon-medical-transcription.com/history_speech_recognition.html
41. D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent dirichlet allocation,” *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.
42. F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain,” *Psychological Review*, vol. 65, pp. 386–408, 1958.
43. simplilearn.com, online; Accessed December 5 and 2020. [Online]. Available: https://www.simplilearn.com/what-is-perceptron-tutorial
44. T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient esitmation of word representations in vector space,” in *ICLR*, 2013.

45. ——, “Distributed representations of words and phrases and their compositionality,” *Advances in Neural Information Processing Systems*, 2013.
46. A. Vaswani, N. Shazeer, N. Parmer, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” in *NeurIPS Proceedings*, 2013.
47. Google and LLC, “Google translate,” online; accessed December 7 and 2020. [Online]. Available: https://support.google.com/translate/answer/2534559?co=GENIE.Platform%3DDesktop&hl=en
48. A. Inc, “Siri,” online; accessed December 7 and 2020. [Online]. Available: https://www.apple.com/siri/
49. Brown and T. et al., “Language models are few-shot learners,” in *NeurIPS Proceedings*, 2020.
50. J. Sommerlad, “Google translate: How does the search giant’s multilingual interpreter actually work?” 2018, online; accessed December 7 and 2020. [Online]. Available: https://www.independent.co.uk/life-style/gadgets-and-tech/news/google-translate-how-work-foreign-languages-interpreter-app-search-engine-a8406131.html
51. A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, “Improving language understanding by generative pre-training,” 2018.
52. A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language models are unsupervised multitask learners,” 2019.
53. OpenAI, “Better language models and their implications,” 2019, online; accessed December 7 and 2020. [Online]. Available: https://openai.com/blog/better-language-models/

Courtni Byun is currently a graduate student at Brigham Young University studying computer science. She earned a B.A. from Brigham Young University in linguistics, with a minor in computer science. She is originally from Missouri.

How Developments in Mathematics led to Modern Techniques in Language Modeling

Jonathan Skaggs
Brigham Young University

***Abstract*—Language Modeling has a long, rich history. The task was originally tackled by Chomsky and other linguists who tried to form a universal grammar for all languages. After, Shannon defined a new type of language model, the statistical language model. Then came the revolution of deep learning and with it the rise of neural language models. After this, came the idea to represent words in the form of a vector space. This history describes the development of the problem of language modeling. The history of the mathematics behind modern ideas in language modeling happened independently until the two ideas were combined together with an ideas called embedding spaces. The mathematical history starts with the initial definition of a vector space by Peano. The idea continues to evolve with the more abstract ideas from Noether called rings, groups, fields and the advent of abstract algebra. Hilbert and Banach continue to develop specific types of vector spaces in the context of abstract algebra. These mathematical ideas form the basis for the incorporation of these ideas into the modern techniques for language modeling. The culmination of both of these ideas leads Mikolov and others to word embeddings and sentence embeddings, the foundation of modern language modeling.**

■ **LANGUAGE MODELING** is an important task in computation and in the broad field of natural language processing. The exact problem formation has varied from application to application and over time. But the task more generally is to represent natural languages (as opposed to computer languages) in a computational form so that a computer can learn to read, write and generally interact well with people. Techniques in language modeling are often implemented in audio processing and natural language processing. With recent advancements, language modeling continues to be used in an increasing number of tasks.

Language modeling, like most important ideas, has a rich history with many people contributing to its ideas. This makes it difficult to include all contributions even in a more comprehensive history. This article attempts to cover the people with the most groundbreaking contributions to the field with specific emphasis on both the history of the concept of language modeling and the mathematical foundations of modern language modeling. It starts with the mathematical background of vector spaces, abstract algebra, and then continues to describe unique cases of vector spaces called Hilbert spaces. After following the evolution of these mathematical foundations, the article continues to explain the history of the concept of language modeling following influen-

Figure 1. Giuseppe Peano was born on 27 August 1858 and died on 20 April 1932. He was an Italian mathematician who work diligently on set theory and logic. He wrote out the axioms of natural numbers which are today called the Peano axioms in his honor. He continued to be a brilliant mathematician and wrote the axioms of a vector space.

tial ideas such as grammars, statistical models, and neural models. Finally, by putting the two histories together, the article will establish the idea of embedding spaces.

Giuseppe Peano

Giuseppe Peano graduated from the University of Turin in 1880. Afterwords, he began his professional career at the same University to work under their chair of calculus, Angelo Genocchi. Shortly after starting his career he published a book on logic where he first used the modern notation for unions and intersections [1].

Peano is most famous for defining the axioms of natural numbers. The axioms are called the Peano axioms in his honor. After defining the Peano axioms, he continued to develop that field of mathematics. In his work, he defined a vector space for the first time. This work would lay the ground work for mathematics of signal processing and language modeling. Although his work with vector spaces was super important, it was not highly recognised until many years later when it was rediscovered by Banach and others.

The following is the definition of a vector space where x and y are elements of a vector space V where 0 is the 0 vector; and c and d are arbitrary scalar values:

Definition 2 Vector Space:

1) $\forall x, y$: $x + y = y + x$
2) $\forall x, y$: $(x + y) + z = x + (y + z)$
3) $\forall x, y$: $0 + x = x + 0 = x$
4) $\forall x, y$: $(-x) + x = x + (-x) = 0$
5) $\forall x$: $0x = 0$
6) $\forall x$: $1x = x$
7) $\forall x, c, d$: $(cd)x = c(dx)$
8) $\forall x, y, c, d$: $c(x + y) = cx + cy$
9) $\forall x, c, d$: $(c + d)x = cx + dx$

Axiom 1-4 describes how a vector space must be closed under addition. It defines that the result of adding two elements of the vector space will also be a member of the vector space. They establish an additive identity, and say that addition is associative and commutative. Axioms 5-7 state that a vector space is closed under scalar multiplication. It defines the existence of a 0 vector and and identity vector. The last 2 axioms state that the distributive property holds. The vector space defined here is the foundation of signal processing that will be used later on to create language modeling systems.

Vector spaces are an interesting idea as presented by Peano, but they lack the generality of modern vector spaces. It may be hard to see at first, but this added generality allows for application to all sorts of problems. This generality is later introduced as abstract algebra by Emmy Noether.

Noether

Emmy Noether was a German mathematician born in 1882. She was an influential mathematician, and according to Albert Einstein she was the most important woman in the history of mathematics. She lived during a time and place where the role of woman in universities was limited. She

Figure 2. Emmy Noether was born in March 1882 and died in April 1935. She studied at the University of Erlangen and began her career in Mathematics there. In 1915 she began work at the University of Göttingen at the request of David Hilbert.

faced lots of gender discrimination but despite this she consistently made great achievements to her field.

She is often accredited with inventing abstract algebra. Different types of algebra have been forming for a long time, one of these being the development of vector algebra, but abstract algebra created an abstract version of all of the different algebras unifying them all into one algebra. This is important because that means that a proof in abstract algebra is a proof in all algebras. Her contributions allowed for the creation of new abstract ideas in linear algebra like groups, rings, and fields.[2] [3] Groups, rings, and fields are fundamental concepts in abstract algebra and consequently are fundamental in the mathematics of signal processing.

Definition 3 A GROUP is a set G which is:

1) Closed under an operation $*$: $\forall x, y \in G$: $x * y \in G$
2) Has an identity element: $\forall x \in G\ \exists e \in G$ where $e * x = x * e = x$
3) Has an inverse element: $\forall x \in G\ \exists y \in G$ where $x * y = y * x = e$ where e is the identity
4) Is associative: $\forall x, y, z \in G$: $x * (y * z) = (x * y) * z$

Definition 4 An ABELIAN is a group with an additional requirement:

1) It commutative: $\forall x, y \in G : x * y = y * x$

Definition 5 A RING is a set R which is closed under two operations $+$ and $*$ and satisfying the following properties:

1) R is an abelian group under $+$
2) Associativity of $*$: $\forall a, b, c \in R$: $a*(b*c) = (a * b) * c$
3) Distributive Properties: $\forall a, b, c \in R$: $a * (b+c) = (a*b)+(a*c)$ and $(b+c)*a = b * a + c * a$

Definition 6 A FIELD is a set F which is closed under two operations $+$ and $*$ such that

1) F is an abelian group under $+$
2) F is an abelian group under $*$
3) Distributive Properties: $\forall a, b, c \in F$: $a * (b + c) = (a * b) + (a * c)$ and $(b + c) * a = b * a + c * a$

Each of these definitions is an important contribution to the understanding of a vector space. A vector space can be defined over a field. This understanding of fields allows for the application of vector spaces on different fields. Examples of some fields are the set of all real numbers $\mathbf{R}^n$, the set of all complex numbers $\mathbf{C}^n$, or the set of all rational numbers $\mathbf{Q}^n$.

Once Noether had defined abstract algebra, the idea of vector spaces was given a larger context in which it could be studied. With this larger context came the rise to two special kinds of vector spaces: Hilbert and Banach spaces.

Figure 3. David Hilbert was a German mathematician born in January 1862 and died February 1943. He is known as one of the most important mathematicians ever. He contributed heavily to the fields of commutative algebra, algebraic number theory, proof theory, and logical mathematics.

Hilbert, and Banach

Shortly after the rise of abstract algebra David Hilbert and later Stefan Banach each introduced a new type of vector space. [4] [5] Today we call these unique vector spaces Hilbert spaces and Banach spaces in their honor.

In order to understand Hilbert and Banach spaces, it is important to understand the purpose of both a norm and an inner product. A norm measures the distance of a vector from the origin. The distance of a norm can be defined in ways other than the common euclidean distance. See definition 7 for an exact definition and some examples.

Definition 7 Norm:

1) $||x|| \geq 0 \forall (x \in S)$
2) $||x|| = 0$ if and only if $x = 0$
3) $||cx|| = |c| ||x||$ where c is a scalar
4) $||x + y|| \leq ||x|| + ||y||$, this is also known as the triangle inequality

Some Common norms are as follows:

1) L_1 Norm: $||x(t)||_1 = \int_a^b |x(t)| dt$
2) L_2 Norm: $||x(t)||_2 = (\int_a^b |x(t)|^2 dt)^{1/2}$
3) L_∞ Norm: $sup_{t \in [a,b]} |x(t)|$ (The supremum function (sup) is similar to the max function but it works better with limits.)
4) L_p Norm: $||x(t)||_p = \int_a^b |x(t)|^2 dt$ for $1 \leq p < \infty$

An inner product is used to measure the similarity between two vectors. If two vectors are orthogonal then the inner product is 0. If the vectors are normalized than the maximum similarity score would be 1. Therefore, the inner product of normalized vectors results in a similarity score of the two vectors between 0-1.

Definition 8 Inner Product:

1) $\langle x, y \rangle = \overline{\langle y, x \rangle}$ where the bar represents the complex conjugate
2) $\langle \alpha x, y \rangle = \alpha \langle x, y \rangle$
3) $\langle x + y, z \rangle = \langle x, z \rangle + \langle y, z \rangle$
4) $\langle x, x \rangle > 0$ if $x \neq 0$ and $\langle x, x \rangle = 0$ if and only if $x = 0$

Definition 9 Complete:

1) A space is Complete if every Cauchy sequence in the space is convergent to a member of the space.

A Banach space is defined as a complete normed vector space, and a Hilbert space is defined as a Banach space with an inner product. Both spaces are complete which put simply means that every converging sequence in the space converges to a number in the space.

Signal Processing Toward Enhanced Language Modeling

The different types of vector spaces and accompanying definitions covered up until now are the culmination of mathematical ideas that form the base of all modern techniques in signal processing.

The mathematics of signal processing is an interesting field because it has seemingly endless

Figure 4. Norm Chomsky is a linguist and cognitive scientist born in December 1928. During Chomsky's career he contributed so much to the field of linguistics that he is often called the father of modern linguistics. He was a professor at the University of Arizona and Massachusetts Institute of Technology.

applications. It has been used in audio processing, image processing, natural language processing, radio processing, radar processing, data compression, and much more.

The remainder of this article will focus on the mostly independent history of the evolution of language modeling. It will then explain how digital signal processing techniques led to the use of vector space representations in language modeling in the form of word and sentence embeddings.

Chomsky

Noam Chomsky was a linguist who discovered the abstract idea of a grammar. He continued his career and established what is today referred to as the Chomsky hierarchy. [6] The basic idea of this hierarchy is to categorize different existing grammars into a hierarchy. The hierarchy says that regular grammars are a subset of context-free grammars which are a subset of context-sensitive grammars which are a subset of recursively enumerable grammars.

Definition 10 A regular grammar G has four components, $G = (N, \Sigma, P, S)$ with the following conditions:

1) N is a nonempty, finite set of non-terminal symbols.
2) Σ is a set of terminal symbols.
3) P is a set of grammar rules that map non-terminal symbols to terminal symbols.
4) $S \in N$ and S is the start symbol.

Each of the grammars in Chomsky's hierarchy have unique applications. Notably, recursively enumerable grammars are comparable Turing machines, or they describe the theoretical potential of computers.

In terms of language modeling, the hierarchy was believed to be extendable from programming languages to natural languages (ie. English, Spanish, ...) as well. Which led Chomsky to the idea of a universal grammar innate in human physiology that allows people to learn natural languages. [7]

Chomsky disliked the idea of statistical language models. He believed that the best form of language models would come from analysing different types of grammars and formally defining a universal grammar for which natural languages could be build. He supposed that all natural languages follow rules from a universal grammar innate to humans. In other words, the human brain is born with a grammar with which they learn to understand language. If this language is discovered then all natural languages could be written in this universal form and could improve the task of language modeling. No such grammar has as of yet been discovered, but there are many people who believe this theory and continue to try to develop this idea.

Shannon

Despite Chomsky's dislike of statistical language models, Claude Shannon formulated the statistical language model problem. [8] The problem is as follows: Given a sequence of words, what is the likelihood of the next word of each word from the vocabulary of all possible words. The purpose of a statistical language model is to

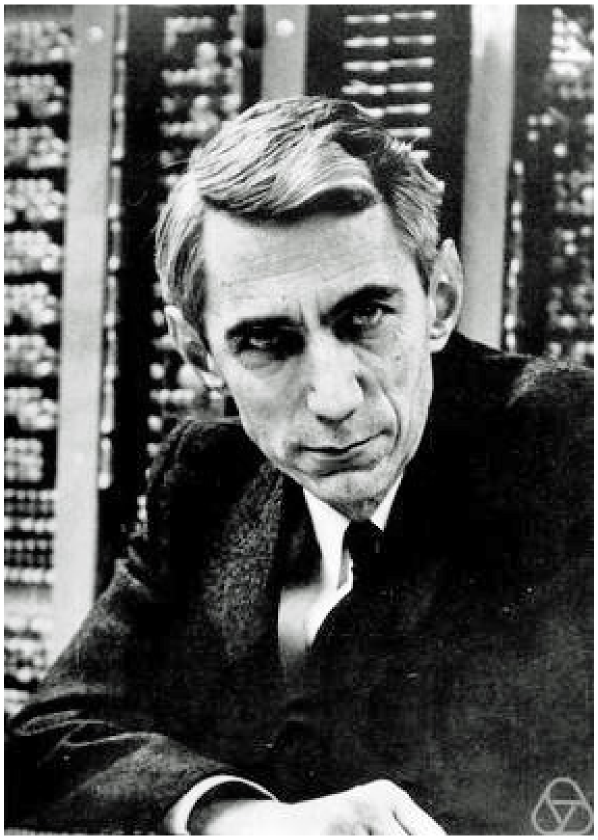


Figure 5. Claude Elwood Shannon was an American mathematician born in April 1916 and died in February 2001. In his career he was a important mathematician and electrical engineer. He is accredited as the founder of information theory. He also contributed heavily to cryptography for national defence during World War II.

model the probability of each word in a sequence $P(w_1, w_2 ... w_n)$. An n-gram is simply a sequence of n words. The simplest n-gram model is called the unigram model. In this case, the sequence of words is actually just a single word. This language model stores the number of times each word shows up and then predicts the probability of each word as the (number of times the word shows up) divided by the (sum of the total number of words). The unigram model finds the probability of a word x as the $P(w_n)$. A bigram model uses the previous word to determine the probability of the current word. This model finds the probability of a word x as the $P(w|w_{n-1})$. Similarly trigrams determine $P(w|w_{n-1}, w_{n-2})$ and 4-grams $P(w|w_{n-1}, w_{n-2}, w_{n-3})$ and so on.

Definition 11 A trigram is defined as :

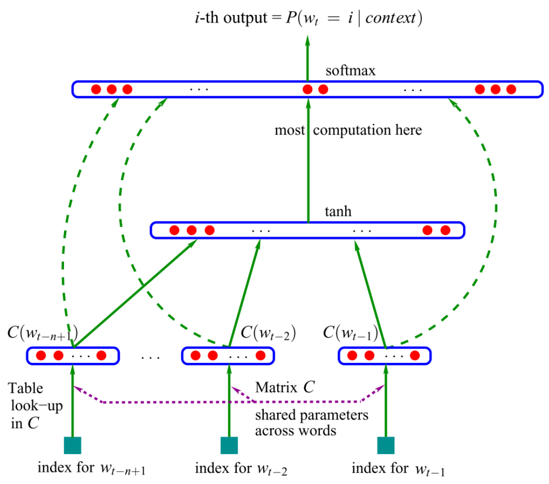


Figure 6. This figure is the network architecture of the first neural language model created by Bengio []. Bengio was born in 1946 in France. He is a professor at the University of Montreal, Canada. He is most well known for his work in deep learning.

$P(w_n|w_{n-1}, w_{n-2}) = \frac{c(w_{n-2}, w_{n-1}, w_n)}{c(w_{n-2}, w_{n-1}, *)}$ where

1) w_n is a particular word in the vocabulary
2) w_{n-2}, and w_{n-1} are the previous two given words in the sequence
3) c is the count function so that c(”I”, ”love”, ”food”) is the number of times the sequence [”I”, ”love”, ”food”] has been seen by our model.
4) $*$ represents all possible words so that c(”I”, ”love”, *) would be the number of times the sequence [”I”, ”love”] was followed by another word.

The n-gram model seems to be a great language model but it also has some difficult technical issues. For instance, it is difficult to deal with out of vocabulary words. The n-gram language model assumes that the same sequences that appeared in the past will appear with the same frequency in the future. This may be generally true of short sequences but the world environment is constantly changing and therefore people are expressing ideas that have never before been expressed. Therefore, as n increases the assumption that n-grams from the past will be displayed in the future is violated. There will never be enough data to overcome this issue.

Bengio

In recent years, the use of deep learning has been on the rise, and with this, deep neural networks have been applied to many different areas that they may not have otherwise. Language modeling has been changed as well by this revolution. The first neural language model was proposed be Yoshua Bengio. This language model takes the previous words as input and outputs the probability that each word in the vocabulary is the next word in the sequence, similar to the n-gram language mode. The difference being that this model does not use probability to solve the problem but instead uses a learned function approximation. (see figure 6 for the model structure)

Collobert and Weston

Ronan Collobert and Jason Weston started to represent language models using embedding spaces.[9] Language modeling for the first time was using the mathematical foundations created by Peano, Noether, Bananch, Hilbert, and others.

Collobert and Weston described a concept known as word embeddings or embedding spaces. The idea behind embedding spaces is to use a neural network to approximate a Hilbert Space in a meaningful way. An embedding space needs a norm to calculate the length of a vector and an inner product to calculate the distance between vectors.

The concept of representing words as a Hilbert Space for the purpose of comparing words was an interesting one. An embedding is complete, normed, and has a defined inner product therefore it is a Hilbert Space. Although setting a language model as a Hilbert space is interesting, it is not without complication. How should the neural network architecture be set up so that it learns a meaningful representation.

As an example, in order to demonstrate the difficulty of this task, imaging taking a dictionary and using a one-hot encoding for each word in the vocabulary. This is a valid representation of the words. We have a Hilbert space, or embedding space, $E \in R^v$ where R is the space of real numbers and v is the size of the vocabulary. Also suppose the inner product is defined as the dot product and the norm is the induced norm. In this space all words are equally far away, the inner product of any two vectors in a one hot encoding is 0. This space is useless. It describes words as being linearly independent and having no relationship with any other words. How then should the encoder be set up? In my opinion, this question defines modern research in deep language modeling.

Figure 7. Thomas Mikolov is a Czech computer scientist. In his career he is known for his work in deep learning and natural language processing. His most notable accomplishment is Word2vec.

Mikolov

To remedy this problem Thomas Mikolov created word2vec.[10] He set up his embedding space under the assumption that words that appear in the same context have similar meaning. This technique was shown to have interesting properties like vector math. The classic example that is all over the internet is if you take the vector for “king”, subtract the vector for “man”, and add the vector for “woman”, then the result is the vector for “queen”, or at least it is close.

In current research there continues to be the debate of how to set up the problem in order to get a meaningful embedding space. One problem with Mikolov’s model for word embeddings is that they capture syntactic meaning instead of semantic meaning. Syntactic meaning is the meaning of the relationship of a word to the words around it; it can be thought of as grammar relationship. Semantic meaning, on the other hand, captures what the word represents in the real world. These two meanings are similar and have a lot of overlap, but arguably the issues found in Word2vec all stem for this difference.

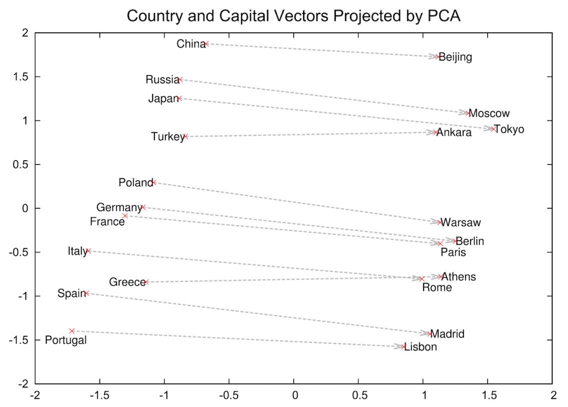


Figure 8. The idea behind Word2vec is to capture the semantic meaning of words using vectors. This figure shows an example of the concept by plotting countries and their capitals using principle component analysis for dimensionality reduction.

Modern Research Directions

After obtaining meaningful semantic word embeddings, or an approximation of this, how are words combined in order to create embeddings of an idea? That is where sentence embeddings come in. The task of sentence embeddings is given an ordered list of word embeddings to create a semantic approximation in a vector space. Like word embeddings, sentence embeddings are also Hilbert spaces and have similar concerns of syntactic meaning vs semantic meaning. Although there is also concerns about what sentence vector math can and should look like in order to achieve the best results.

There continues to be a lot of research into both word and sentence embeddings. [11] [12] [13] [14] [15] [16] [17] The most notable current state of the art language models are Bert and GPT-2. [18] [19] Each of these language models use the same underling concepts to form a sentence embedding. It will be interesting to see how this idea continues to evolve and progress.

REFERENCES

1. P. Giuseppe, *Calcolo geometrico secondo l'Ausdehnungslehre di H. Grassmann: preceduto dalla operazioni della logica deduttiva*, vol. 3. Fratelli Bocca, 1888.
2. E. Noether, "Idealtheorie in ringbereichen," *Mathematische Annalen*, vol. 83, no. 1-2, pp. 24–66, 1921.
3. B. L. Van der Waerden, E. Artin, and E. Noether, *Moderne algebra*, vol. 31950. Springer, 1950.
4. D. Hilbert, "Beweis für die darstellbarkeit der ganzen zahlen durch eine feste anzahl n ter potenzen (waringsches problem)," *Mathematische Annalen*, vol. 67, no. 3, pp. 281–300, 1909.
5. N. Bourbaki, "Sur certains espaces vectoriels topologiques," in *Annales de l'institut Fourier*, vol. 2, pp. 5–16, 1950.
6. N. Chomsky, "Three models for the description of language," *IRE Transactions on information theory*, vol. 2, no. 3, pp. 113–124, 1956.
7. N. Chomsky, "The galilean challenge: Architecture and evolution of language," in *Journal of Physics: Conference Series*, vol. 880, 2017.
8. C. E. Shannon, "A mathematical theory of communication," *ACM SIGMOBILE mobile computing and communications review*, vol. 5, no. 1, pp. 3–55, 2001.
9. R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of machine learning research*, vol. 12, no. ARTICLE, pp. 2493–2537, 2011.
10. T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, pp. 3111–3119, 2013.
11. J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.
12. M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," *arXiv preprint arXiv:1802.05365*, 2018.
13. Y. Sun, S. Wang, Y. Li, S. Feng, X. Chen, H. Zhang, X. Tian, D. Zhu, H. Tian, and H. Wu, "Ernie: Enhanced representation through knowledge integration," *arXiv preprint arXiv:1904.09223*, 2019.
14. A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.
15. A. M. Dai and Q. V. Le, "Semi-supervised sequence learning," in *Advances in neural information processing systems*, pp. 3079–3087, 2015.
16. J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," *arXiv preprint arXiv:1801.06146*, 2018.

17. D. Cer, Y. Yang, S.-y. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, *et al.*, "Universal sentence encoder," *arXiv preprint arXiv:1803.11175*, 2018.
18. A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.
19. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

Jonathan Skaggs is a PhD student in Computer Science at Brigham Young University. His research involves incorporating models of persuasion into the architecture of neural language models.

Machine Translation

J. S. Rydalch
Brigham Young University

***Abstract*—This paper covers the history of different approaches taken in machine translation along with brief technical overviews for each approach. The first main group of translation approaches is rule-based machine translation. This includes the early direct translation approach used by initial researchers, the more abstract interlingua approach, and the transfer approach that utilizes advances made in the direct and interlingua approaches. The paper then covers the departure from rule-based systems in the form of example based machine translation, statistical machine translation, and neural machine translation.**

■ **THE DEVELOPMENT OF MACHINE TRANSLATION** has had a large number of researchers and contributors, ranging from diverse backgrounds. Influenced by ideas from information theory, computer science, statistics, linguistics, and deep learning, it is a multi-faceted area of research. Initially spurred on by international interests and focused on Russian and English translation, it has shifted over time to incorporate every living language on the planet, and some dead languages as well. At its core, machine translation is concerned with the problem of transforming information from one language to another. As technology as a whole has grown more sophisticated, some machine translation problems focus on the spoken word and real-time translation, but machine translation has historically been purely text based. A plethora of methods for machine translation have been proposed, and these approaches have been iterated upon and combined to create the existing systems in use today.

ORIGINS

Aspects of the idea of machine translation have existed and been proposed by great thinkers for hundreds of years. One of the major approaches finds its root in the musings of Leibniz and his universal characteristic. This was Leibniz's dream of a series of symbols or an alphabet in which the entire scope of human thought could be represented [1]. With such an approach, one could simply create a methodology to transform one language into this universal language and then have a separate process to generate text in another language. While such an approach was initially considered, attempts to create such a system were not seriously undertaken at first.

As one of the original uses for mechanical computers was cryptography, the problem of translating information from one form to another was already associated with computation. However, the beginning of research into machine translation stems from a memorandum sent by Warren Weaver in 1949 [2]. This memo, which was sent to some 200 of Weaver's colleagues, recounts a few experiences and observations Weaver had concerning cryptography and how similar approaches could be used to translate from plain text of one language to another. It also covers some of the earliest experiments undertaken in machine translation by Richard H. Richens and Andrew D. Booth in England using punch cards and simple word for word substitution, which they expanded upon and published at a later date [3]. Interestingly, Weaver also explores concepts of statistical machine translation in this memo, which while not initially experimented with, re-emerged in the late 1980s and developed into a major area of study.

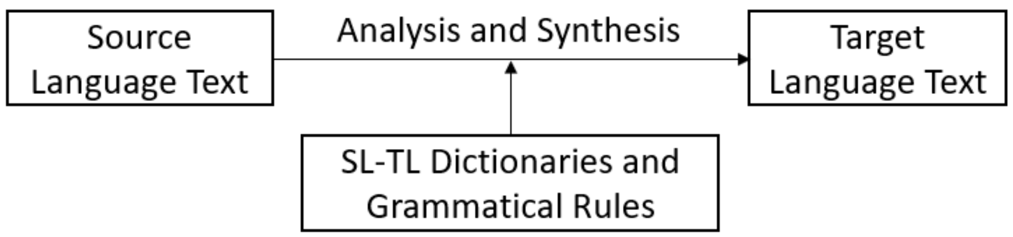


Figure 1. Direct Machine Translation System. In direct machine translation, individual words from the source language are substituted for the equivalent target language word based on the included dictionaries. Then, the substituted words are ordered and manipulated according to any rules, grammatical or otherwise, included in the program.

EARLY EXPERIMENTS

The title of the first researcher in the field of machine translation is often given to Yehoshua Bar-Hillel. A veteran of World War II, Bar-Hillel received a PhD in Philosophy from the Hebrew University and completed a post-doctorate at the University of Chicago in 1950. Following that post-doctorate, he joined IBM in May of 1951 and began work on machine translation. Bar-Hillel organized the first International Conference on Machine Translation, and virtually every active researcher in the field attended [4]. Even this early on in the history of machine translation, it was considered impossible to generate high quality translations purely automatically. Most researchers agreed that human intervention at some stage of the process would be necessary. Not all considered that this would always be the case, but the majority opinion was that mechanical translation could not completely replace human effort. One attendee of note was Léon Dostert of Georgetown University. He suggested that a public demonstration of machine translation could spark interest and secure funding for research into machine translation.

Then in 1954, Léon Dostert in collaboration with IBM set up a public demonstration in an attempt to prove the viability of machine translation. This demonstration utilized a small dictionary set of words and hand selected examples of Russian sentences that the system could handle. This demonstration was successful enough to spur on the creation of multiple other machine translation research projects, both in various institutions in the USA and throughout the world.

In the same year, the first machine translation journal was founded by William Locke and by Victor Yngve, who had succeeded Bar-Hillel at MIT in 1953. Also in this year, the first doctoral thesis in machine translation was completed by Anthony G. Oettinger. The years 1954 and 1955 saw the foundation of a group in Cambridge, England, under Margaret Masterman, a group in Milan under Silvio Ceccato, the first Russian groups at the Institute of Precise Mechanics and Computer Technology, the Institute of Applied Mathematics, Leningrad University, etc. and the start of various Chinese and Japanese projects. And in 1955 the first MT book appeared, a collection edited by Locke and Booth (1955), including Weaver's 1949 memorandum, Booth and Richens' experiments, some papers given at the 1952 conference, and other contributions from Bar-Hillel, Dostert, Oettinger, Reifler, and Yngve [4].

The Direct Translation Approach

The approach utilized in the Georgetown University demonstration and in many other early systems is often called the 'direct translation' approach. This approach is designed for a specific pair of languages and works in only one direction. It translates from the source language (SL) into the target language (TL). Each new pair of languages requires a new system to be developed. The systems in question usually consisted of two parts: a large dictionary mapping words from the source language to the target language, and a large program in charge of text generation and analysis.

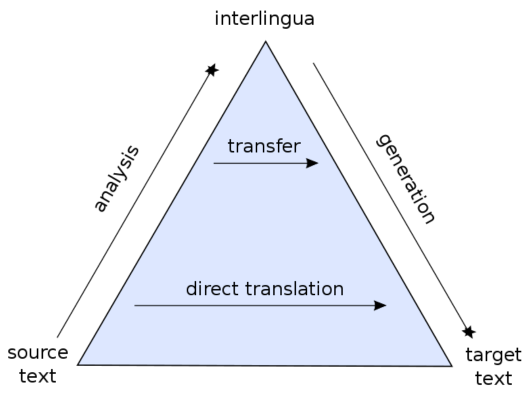


Figure 2. A Vauquois triangle, used to represent the idea behind different levels of translation. As approaches move higher on the triangle, they involve more abstraction and analysis. For example, the interlingua approach involves abstracting the source text entirely, while direct translation involves little analysis or abstraction.

The development of these direct translation programs, later called the 'brute force' method by Paul Garvin (a prominent researcher during the 1960s) [5], consisted of developing a program to translate a specific corpus from the source language into the target language and then testing the program on a new corpus and making additions and adjustments until the program could translate the new corpus. This process would be repeated on newer and larger corpora, gradually creating an incredibly complicated program with small exceptions and grammatical rules appearing scattered throughout the program. Without a structured approach, many of these programs were nearly impossible to iterate on after a certain point and were rarely modified after deployment.

The issue with these methods was that virtually every effort for one program was not usable in other programs. For example, creating a program to translate from German to English, the sentence "Das Papier ist weiß" into the English equivalent "The paper is white" can mostly be accomplished via dictionary substitution. "Das" becomes "The" and a rule would be included for post-processing to capitalize the first letter of the sentence. The next three words translate directly. This is a toy example however, and more complicated words and phrases could cause issues in the choice of English synonyms. Even if the German to English translation program worked flawlessly, it could not simply be turned around to produce English to German translations. For example, German has masculine, feminine, and neuter noun cases. When translating from German to English, "der", "die", and "das" all translate to "the." However, translating from English to German, a special rule must be involved that looks at the noun following "the" and determines its gender. In the direct translation approach, researchers and programmers would repeatedly tackle issues by adding additional rules or expanding the translation dictionaries, but it is readily apparent how such an approach would quickly lead to an incomprehensible mess of proprietary rules and quick fixes.

These direct approaches did not involve linguistics in any major capacity, and while initial progress had seemed swift, the National Science Foundation created the Automatic Language Processing Advisory Committee (ALPAC) to review the efforts and progress being made on machine translation. They released a crippling report in 1966 [5] that described machine translation as slower, less accurate and twice as expensive as human translation, recommending no further investment. ALPAC recommended that instead of attempting to create fully automatic high quality translation systems, research should focus on machine aids for translators, such as automatic dictionaries. This report severely curtailed the machine translation research occurring in the USA for over a decade, with many projects being shut down entirely. This was a dark period for machine translation, with many researchers and professionals seeing machine translation as a failure.

DEVELOPING ALTERNATE METHODS

Following the ALPAC report in 1966, the remaining groups continuing to research machine translation began to focus on different approaches. Whereas most of the methods so far had focused on direct translation, with little thought given to linguistic aspects, this period developed the interlingua approach. The basic idea behind the interlingua approach is that instead of translating one language to another via

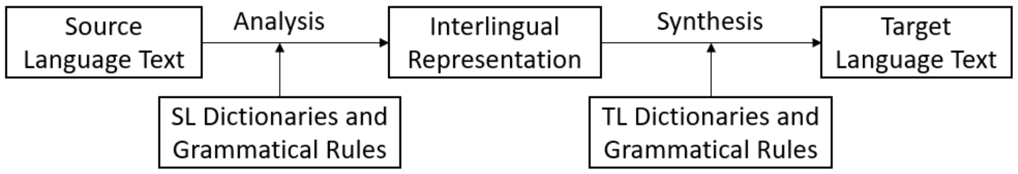

Source Language Text
Analysis
Interlingual Representation
Synthesis
Target Language Text
SL Dictionaries and Grammatical Rules
TL Dictionaries and Grammatical Rules

Figure 3. Interlingual Machine Translation System. Interlingual machine translation systems attempt to first convert a source language text into an abstract interlingua. From this representation, text in the target language can be created.

word substitution and corpus specific grammatical rules, the SL would be translated into an abstract interlingua representation. From this interlingua representation, text in the TL could be generated. This approach hearkens back to Leibniz's universal characteristic idea, and had many appealing properties. If a standardized interlingua could be created, then independent groups could create encoders and decoders that transformed a particular language into interlingua and interlingua into their particular language. So for a multilingual translation system involving *n* languages, only *2n* programs are needed, as opposed to the direct translation approach, which would need *n(n-1)* language pairs to achieve the same results [6]. On a worldwide scale, a system could potentially be developed to allow a series of encoders and decoders to translate a text in any source language into any target language, simply by collecting and utilizing the various encoders and decoders developed by independent groups. While this grand vision of interlingua translation was never fully realized, the research into this approach revealed important new approaches and methods of thinking about machine translation.

The USA was not the only region to research machine translation. Due to their bicultural setup, Canada had need for translations from French to English and English to French. Europe too was invested in creating systems that could translate between the various languages in use by European powers. France had created 'Centre d'Etudes pour la Traduction Automatique' (CETA), an organization focused on machine translation. This had split into CETA-P in Paris and CETA-G in Grenoble. Bernard Vauquois's group at Grenoble University in France developed a rudimentary interlingua translation method from 1960 to 1971 [7]. Their approach was not a pure interlingua. It utilized syntactic analysis to generate an abstract representation of the relationships between the predicates and arguments of the text. Actual words and phrases were translated via a bilingual transfer, similar to the direct translation approach. However, after converting the words into the TL, the abstract representation was used to order these words and phrases and generate TL sentences based off of that structure.

During this period, other groups also adopted and experimented with interlingua approaches, including groups at the Linguistic Research Center (LRC) in Texas, and Mel'chuk, a prominent Soviet Union researcher. However, by the mid 1970's, the interlingua approach was still wracked with major issues that raised concerns about its potential. At any stage of the process, if a proper analysis fails, the entire translation fails as well. This necessitated the creation of incredibly robust parsers, which led to inefficiencies as even for simple translations the parsers would need to consider edge cases and complicated potential meanings. Another argument against interlingua translation was the inherent loss of information that occurs when encoding into interlingua. Direct translation can rely on the SL sentence for the entire translation process, allowing it to glean information from the SL that would help in determining proper forms and sentence structures for the TL. Without a lossless interlingua, information would be lost upon the encoding into interlingua that the decoders could not use, leading to complications that other approaches would not have [6].

REVITALIZATION AND THE TRANSFER APPROACH

As a compromise between the pure abstraction of interlingua approach and the nonexistent

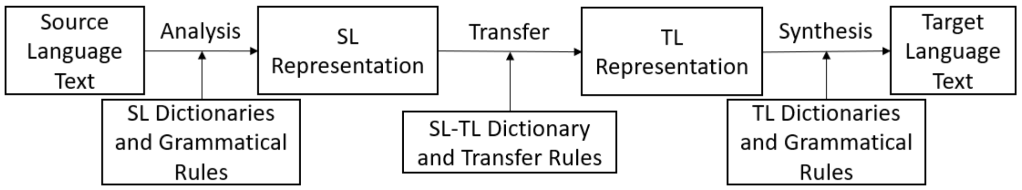

Source Language Text
Analysis
SL Representation
Transfer
TL Representation
Synthesis
Target Language Text
SL Dictionaries and Grammatical Rules
SL-TL Dictionary and Transfer Rules
TL Dictionaries and Grammatical Rules

Figure 4. Transfer Machine Translation System. Transfer based systems are broken into three steps. First is the analysis of the source language to generate an abstract representation unique to the source language. This representation is then transferred into a similar abstract representation unique to the target language. From the target language representation, text in the target language is synthesized.

abstraction of the direct approach, researchers developed the transfer approach. The main idea behind the transfer approach is similar to interlingua, but split into two proprietary abstractions for both the source language and target language. The SL is analyzed and transformed into an abstract SL representation. This representation is then 'transferred' and converted into an abstract TL representation. From that TL representation, text in the TL is generated. This approach, while less ambitious and lacking some of the potential benefits of interlingua, was much more attainable. The transfer approach had the advantage of using linguistic analysis, but without the same degree of loss of information present in interlingua approaches.

From the mid 1970s until the end of the 1980s, much of the research into machine translation was focused on transfer machine translation. The Grenoble group, eventually began development of its Ariane system. This system is often regarded as the "paradigm of the 'second generation' of linguistics-based transfer systems." [6]. While the Ariane system was never deployed in practice, it was incredibly influential for this period, and many of the systems developed worldwide share a striking similarity to the concepts explored with Ariane.

Japan saw significant interest and investment in machine translation in the 1980. One of the lead researchers in Japan at the time was Makoto Nagao, who led the Mu project at the University of Kyoto. The Mu project incorporated the three-stage transfer approach that most of the networks of the period used. However, the significant iterations in the form of case grammar analysis and dependency tree representations, along with the development of GRADE, a system used to describe linguistic grammars (GRAmmar DEscriber), cemented its foundational position for machine translation in Japan. The system developed by the Mu project was focused on translating technical papers from Japanese, and their 1986 prototype was used by the Japanese Information Center for Science and Technology to translate abstracts [4].

One of the more famous machine translation efforts is know as SYSTRAN. SYSTRAN was founded in 1968 by Peter Toma and was not nearly as affected by the ALPAC report of 1966. Originally started as a direct translation effort based off of the work done for the Georgetown demonstration, SYSTRAN quickly became a main player in the realm of machine translation [5]. SYSTRAN during this period began to incorporate design aspects of transfer machine translation systems while retaining much of its traditional direct translation approach. Their paired language translations were effective enough to secure deals with multiple European countries for systems to translate between Latin based languages [4].

Despite the numerous advances created over the years in regards to rule-based machine translation (RBMT), it still had glaring weaknesses. RBMT systems needed to be tuned to a specific area of language, such as engineering or technical and would usually perform poorly on texts outside of its area of expertise. Efforts to create a more generalized approach to RBMT systems that could handle texts from any genre were often met with failure. The series of rules needed for quality translations expanded as the scope of the system did. Additionally, the problem of non-transferable work for one language to another had still not been solved. Transfer systems did much to help with this issue, but every pair of languages required a mostly custom set of transfer rules. This was both expensive and prone to error, with every new system requiring a large amount of overhead. Research was looking into other alternatives that could potentially be easier to set up and maintain than the traditional RBMT systems.

A DEPARTURE FROM RULE-BASED SYSTEMS

While both the ‘first’ and ‘second’ generation of machine translation relied on rule-based translation, translation that algorithmically analyzed or generated text from a SL to a TL, in the late 1980s, new approaches were being explored and rapidly gained in popularity. These include example based machine translation and statistical machine translation. It is important to note that during this period machine translation had been slowly shifting into amalgamated methods, with most professional approaches blending a number of techniques. So while these new systems strayed from the original core of rule-based translation, it was not uncommon to use previously developed methods in combination with newer methods.

Example Based Machine Translation

As early as 1981, a Japanese researcher Makoto Nagao had been working on an alternative method of machine translation. In 1984 Nagao proposed a method for machine translation based off of the manner that human translators followed [8]. Instead of trying to create abstract representations for sentences, human translators often break a body of text into phrases that they then translate directly. Then, the rest of the work relies on developing a translation that connected the phrases together. Nagao reasoned that machines could do something similar with a large set of example translations. Starting from a text professionally translated into two languages, the texts could be aligned and broken down into phrases by utilizing rule-based approaches or statistical analysis. A database of phrases could then be generated, with larger corpora creating richer databases. Once the database was established, a SL text could be broken down into phrases that either exactly matched or approximated existing phrases in the database and the corresponding translation could be substituted. This method would eventually come to be known as example based machine translation.

This methodology took hold in Japan, and much of the research into machine translation in Japan during the late 1980s onward focused on example based machine translation. It had several advantages over the traditional rule-based approaches. After decades of research, it was readily apparent that rule-based systems required a large amount of investment and experimentation to develop an appropriate set of grammatical and linguistic rules. Additionally, each new language required unique rules, and while generalized approaches could be applied across the board, much of the work done on rule-based systems was only applicable to the specific language pair addressed. Example based machine translation on the other hand could be improved independently and the methods used could be incorporated into an existing project more readily. Additionally, since example based machine translation used professionally translated examples, large SL phrases that matched with examples in the database would result in professional quality translations [4].

However, example based machine translation did have several drawbacks. While it works well for phrases, connecting phrases together was a non-trivial problem, and two similar phrases in the SL could match to two different phrases in the database, often leading to inconsistency in how some subjects were discussed. Additionally, when faced with a phrase vastly different from anything in the database, example based machine translation does not perform well and can generate very poor quality translations.

Statistical machine translation

In the late 1980s, machine translation researchers began to delve into some of the older ideas of machine translation, going back as far as its origins. Cryptography is heavily related to machine translation, with the main difference being that the ‘source language’ is a unique language. Methods developed by cryptographers had been initially considered as early as Weaver’s memo, but the majority of the work done on machine translation focused on the advantage translation had over cryptography: bilingual resources. However, a second look at the methods developed for cryptography yielded results. Al-Kindi, a famous ninth century Arab mathematician, had described a method known as frequency analysis. In cryptography it is used to crack character substitution encodings by looking at the frequency of characters in the encoded message and comparing it to the frequencies of non-encoded language. This method is able to break single-character substitutions without understanding the encoding method. When applied to machine translation, it could mean that purely statistical processes could "decode" language from a SL into a TL. Of course, the actual method used is significantly altered from frequency analysis, but the core of the idea is similar.

In 1989, a group from IBM published the results of a machine translation system based on statistical methods instead of rule-based analyses. While machine translation had used statistical methods before, they had largely been auxillary. This newer method had statistical analysis as the core of the entire translation. The essence of the method was first to align phrases, word groups, and individual words of the parallel texts, and then to calculate the probabilities that any one word in a sentence of one language corresponds to a word or words in the translated sentence with which it is aligned in the other language (a 'translation model'). The outputs were then checked and rearranged according to word-to-word transition frequencies in the target language, derived from the corpus of bilingual texts (a 'language model') [4]. To the surprise of both linguists and machine translation researchers, the results of this statistical method often exactly matched professionally translated SL text, or offered alternative legitimate translations of SL text when it did not match exactly.

Statistical machine translation had its fair share of issues however. Unlike rule based systems, specific translation errors were nearly impossible to fix, as the translation process was more opaque. Additionally, the creation of appropriate corpora could be a prohibitive cost for more obscure languages, as the corpora needed to be of professional quality and plentiful.

However, despite these drawbacks and others, statistical machine learning soon came to the forefront as one of the premier methods of machine translation. Many large corporations, some with long histories in machine translation, began developing statistical machine translation systems. SYSTRAN, which historically had used a rule-based machine translation system, released a statistical machine translation service in 2010. Google Translate was launched in 2006 and utilized statistical machine translation almost exclusively. Microsoft Translator also utilized statistical methods in its translation service, which started as early as 2000.

While initially different approaches to corpus based machine translation, statistical machine translation and example based machine translation gradually blended together. Statistical machine translation was seen as the best method for corpus-based machine translation, but as it expanded into phrase-based translation it began drawing more on concepts originally developed for example based translation. From around 2000 to 2016, much of the research into machine translation was focused on corpus-based machine translation, with an emphasis on statistical machine translation. Most of the large machine translation corporations either incorporated or exclusively used statistical machine translation until around 2016.

NEURAL MACHINE TRANSLATION

Building off of the more empirical approach to machine translation, the idea to use neural networks for machine translation surfaced in late 2013. Due to the focus on corpus-based approaches, large swaths of bilingual training data were available to train the neural networks. In the following year, a multitude of papers were released concerning the idea of neural machine translation. Additionally, Dzmitry Bahdanau, KyungHyun Cho, and Yoshua Bengio released the paper "Neural Machine Translation by Jointly Learning to Align and Translate" in 2014 [9]. This paper was a turning point in the machine translation community. From 2015 onward, neural machine translation research exploded.

The approaches to neural machine translation are varied. The original proposition for neural machine translation focused on using neural networks as simple encoders and decoders. In the original model, the encoder would have as input a sentence in the source language, and would encode it into a fixed length vector [9]. This vector would then be fed into a decoder network that would generate text in the target language. This approach had difficulties, and following the Bahdanau paper, previous ideas from machine translation were incorporated into the design of the networks. These ideas pulled from statistical machine translation, example based machine translation, and the various rule-based machine translation methods. This area of research is still in rapid development, with papers and results of experiments being published regularly. With every new advancement in deep learning and neural network design, machine translation quickly adapts and implements to accommodate the change.

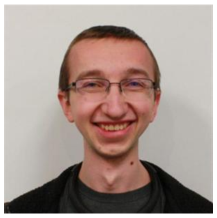
Neural machine translation at this point had demonstrated itself to be a powerful tool, quickly becoming comparable to and often better than the statistical machine translation approaches utilized. As a result of this, multiple large corporations with machine translation efforts announced a transition toward neural machine learning. This list includes SYSTRAN, Microsoft, and Google. For example, in 2016 Google officially switched its main translation service, Google Translate, from a statistical method employed in 2007 to a neural network based approach. While the previous statistical method utilized by Google translated text into English as an intermediary language, the networks employed for the Google Neural Machine Translation system (GNMT) translate directly from the SL to the TL.

CONCLUSION

Machine translation is a field that has existed virtually since the beginning of the development of computer science as a discipline. In its early stages it focused on direct results and largely ignored linguistic interpretation. After the perceived failure of this approach, researchers shifted focus to a rule-based system that borrowed concepts from linguistics. Following this it shifted once again to result-based methods with statistical and example based machine translation, and has now involved the Pandora's Box that is deep learning and neural networks. However, while the methods and approaches have changed radically, and many groups blend multiple styles together to achieve greater results, the main goal of machine translation is unchanged. Machine translation aims to make the concept of a 'language barrier' something to be considered only in the history books.

REFERENCES

1. M. Davis, *The Universal Computer.* CRC Press, 2018.
2. W. Weaver, "Translation." 1949.
3. R. Richens and A. Booth, "Some methods of mechanized translation," pp. 24–46, 1955.
4. W. J. Hutchins, "Machine translation: a concise history," 2006.
5. W. Hutchins, "The evolution of machine translation systems," *Practical experience of machine translation*, pp. 21–37, 1982.
6. W. Hutchins, "Machine translation: A brief history," *Concise history of the language sciences: from the Sumerians to the Cognitivists*, pp. 431–445, 1995.
7. B. Vauquois and C. Boitet, "Automated translation at grenoble university," *Comput. Linguist.*, vol. 11, p. 28–36, Jan. 1985.
8. M. Nagao, "A framework of a mechanical translation between japanese and english by analogy principle," *Artificial and human intelligence*, pp. 351–354, 1984.
9. D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2014.



Jared S Rydalch is a PhD student at Brigham Young University, where he earned a B.S. in Computer Science. He works on problems with computational creativity and deep learning. In particular, he is interested in using neural networks to generate supplemental material for human teaching and learning.

The Emergence of High-Level Programming Languages

Melanie Jensen
Brigham Young University

***Abstract*—Computers use machine code internally, but almost all contemporary computer programming is carried out in high-level languages based on English-language words. This paper endeavors to show how compiled or interpreted high-level languages came to be invented, accepted, and standardized. I begin with the handwritten tables of inputs and outputs for Charles Babbage's theoretical Analytical Engine and conclude with the creation of COBOL, FORTRAN, and ALGOL.**

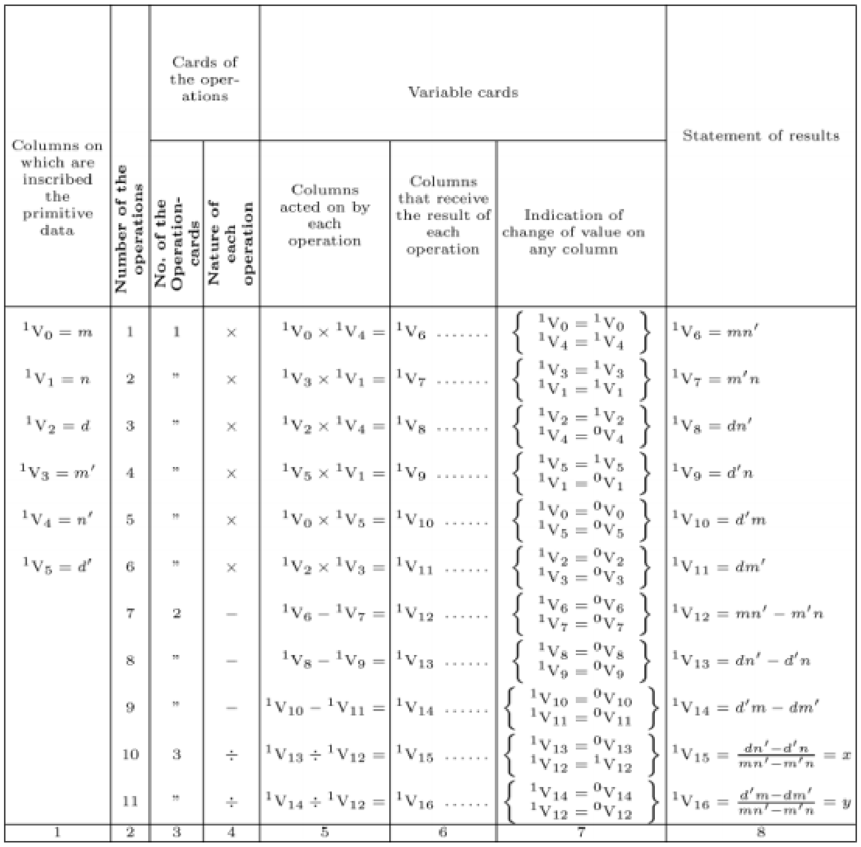
■ **A PROGRAMMING LANGUAGE** is a point between human language and machine code positioned such that it is a reasonably sized step from each. For the gap between human language and programming language, the gap must be small enough for a large proportion of the human population to be able to manage the translation, or in other words, to write code. The smaller the gap, or the closer the programming language is to human language, the larger that proportion can be. For the gap between programming language and machine language, the gap must be small enough that some machine can manage the translation, though it need not be the same machine on which the code will run. The smarter our machines become, the larger that gap can be. Over time, the size of both gaps has varied, but the efforts of visionary human beings and the development of more capable machines has resulted in the trend of programming languages becoming more friendly toward human beings.

This paper will examine the evolution of our communication with computing machines, including those that are mechanical and electromechanical. It is organized mostly chronologically, with separate sections for each programming language. Sections will be named after the programming language they explore, if it has a name, or the name of the hardware on which it is used if it does not. Most sections will include an example of the programming language.

THE ANALYTICAL ENGINE

In the early nineteenth century, tables of logarithms and trigonometry were widely used in calculations relevant to navigation, engineering, science, and finance. However, the tables were tedious to create and sometimes inaccurate. Charles Babbage believed they could be produced quickly and accurately if done mechanically, and he designed a "Difference Engine" to do it. In 1823, he convinced the British government to provide funding for its construction, but the work was slow. The first prototype was not completed until 1832. The following year, 17-year-old Ada Lovelace attended a party at Babbage's home with her mother, and they were invited to see a demonstration of the prototype, which could raise numbers to second and third powers and find the root of a quadratic equation. Later that year, Babbage's engineer quit, keeping all of the plans for the Difference Engine, including the ones drawn by Babbage. [1]

Babbage turned his attention to a better invention, an "Analytical Engine." It would be capable of many different operations, which could



Columns on which are inscribed the primitive data	Number of the operations	Cards of the operations		Variable cards			Statement of results
		No. of the Operation-cards	Nature of each operation	Columns acted on by each operation	Columns that receive the result of each operation	Indication of change of value on any column	
${}^1V_0 = m$	1	1	$\times$	${}^1V_0 \times {}^1V_4 =$	${}^1V_6$	$\left\{ \begin{matrix} {}^1V_0 = {}^1V_0 \\ {}^1V_4 = {}^1V_4 \end{matrix} \right\}$	${}^1V_6 = mn'$
${}^1V_1 = n$	2	"	$\times$	${}^1V_3 \times {}^1V_1 =$	${}^1V_7$	$\left\{ \begin{matrix} {}^1V_3 = {}^1V_3 \\ {}^1V_1 = {}^1V_1 \end{matrix} \right\}$	${}^1V_7 = m'n$
${}^1V_2 = d$	3	"	$\times$	${}^1V_2 \times {}^1V_4 =$	${}^1V_8$	$\left\{ \begin{matrix} {}^1V_2 = {}^1V_2 \\ {}^1V_4 = {}^0V_4 \end{matrix} \right\}$	${}^1V_8 = dn'$
${}^1V_3 = m'$	4	"	$\times$	${}^1V_5 \times {}^1V_1 =$	${}^1V_9$	$\left\{ \begin{matrix} {}^1V_5 = {}^1V_5 \\ {}^1V_1 = {}^0V_1 \end{matrix} \right\}$	${}^1V_9 = d'n$
${}^1V_4 = n'$	5	"	$\times$	${}^1V_0 \times {}^1V_5 =$	${}^1V_{10}$	$\left\{ \begin{matrix} {}^1V_0 = {}^0V_0 \\ {}^1V_5 = {}^0V_5 \end{matrix} \right\}$	${}^1V_{10} = d'm$
${}^1V_5 = d'$	6	"	$\times$	${}^1V_2 \times {}^1V_3 =$	${}^1V_{11}$	$\left\{ \begin{matrix} {}^1V_2 = {}^0V_2 \\ {}^1V_3 = {}^0V_3 \end{matrix} \right\}$	${}^1V_{11} = dm'$
	7	2	$-$	${}^1V_6 - {}^1V_7 =$	${}^1V_{12}$	$\left\{ \begin{matrix} {}^1V_6 = {}^0V_6 \\ {}^1V_7 = {}^0V_7 \end{matrix} \right\}$	${}^1V_{12} = mn' - m'n$
	8	"	$-$	${}^1V_8 - {}^1V_9 =$	${}^1V_{13}$	$\left\{ \begin{matrix} {}^1V_8 = {}^0V_8 \\ {}^1V_9 = {}^0V_9 \end{matrix} \right\}$	${}^1V_{13} = dn' - d'n$
	9	"	$-$	${}^1V_{10} - {}^1V_{11} =$	${}^1V_{14}$	$\left\{ \begin{matrix} {}^1V_{10} = {}^0V_{10} \\ {}^1V_{11} = {}^0V_{11} \end{matrix} \right\}$	${}^1V_{14} = d'm - dm'$
	10	3	$\div$	${}^1V_{13} \div {}^1V_{12} =$	${}^1V_{15}$	$\left\{ \begin{matrix} {}^1V_{13} = {}^0V_{13} \\ {}^1V_{12} = {}^1V_{12} \end{matrix} \right\}$	${}^1V_{15} = \frac{dn'-d'n}{mn'-m'n} = x$
	11	"	$\div$	${}^1V_{14} \div {}^1V_{12} =$	${}^1V_{16}$	$\left\{ \begin{matrix} {}^1V_{14} = {}^0V_{14} \\ {}^1V_{12} = {}^0V_{12} \end{matrix} \right\}$	${}^1V_{16} = \frac{d'm-dm'}{mn'-m'n} = y$
1	2	3	4	5	6	7	8

Figure 1. Menabrea's instructions for solving a system of equations. The first column indicates input data that will be set on the "columns," ie. variables, of the Analytical Engine before the program runs. The fourth column indicates which operation will be run at each step, the fifth column indicates the operand variables, and the sixth column is the output variables. Columns 4-6 would be input using punched cards. The last two columns are comments for the reader. Menabrea's instructions demonstrate the power of storing values in variables and reusing them. (See row 11, where he uses the same stored values to calculate y as he used for x.)

be done in an arbitrary sequence controlled by punched cards, after the fashion of the Jacquard loom. Columns of rotating disks would store decimal numbers which could be used as operands or results. Though Babbage was unable to get funding for the Analytical Engine, he was invited to lecture on it in Turin, Italy, in 1840. Luigi Menabrea, a 30-year-old army engineer who later became prime minister of Italy, took notes on the presentation and published a paper in French. Lovelace, who had remained friends with Babbage, translated the paper into English, adding notes of her own that were longer than the original publication. She included a sequence of 25 operations which would compute Bernoulli numbers. Her programming language, in this case, was a table indicating the operations to be used and the columns (data registers) they would act upon. Other columns in the table contained comments and structural information to help the reader. The same format was used by Menabrea in his paper. [1]

Though Ada Lovelace is generally given credit for being the first programmer, due to her program for Bernoulli numbers, the fact is that Menabrea and Babbage created similar lists of instructions, though not as complicated. The intent of this paper, however, is to explore the programming language and not to answer the question of who deserves the most credit for using it. Figure 1 shows Menabrea's program to solve the system of equations

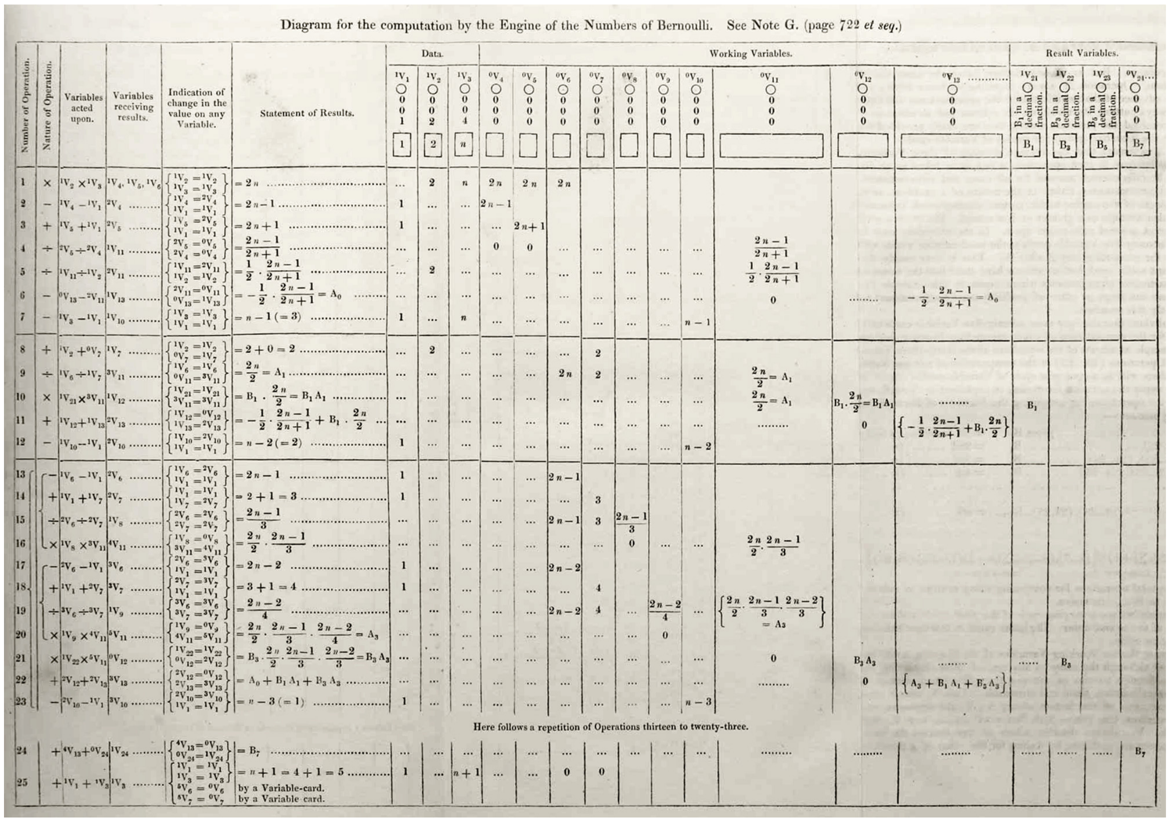


Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 *et seq.*)

Number of Operation.	Nature of Operation.	Variables acted upon.	Variables receiving results.	Indication of change in the value on any Variable.	Statement of Results.	Data.			Working Variables.										Result Variables.			
						$^1V_1$ ○ 0 0 1 $\boxed{1}$	$^1V_2$ ○ 0 0 2 $\boxed{2}$	$^1V_3$ ○ 0 0 4 $\boxed{n}$	$^0V_4$ ○ 0 0 0	$^0V_5$ ○ 0 0 0	$^0V_6$ ○ 0 0 0	$^0V_7$ ○ 0 0 0	$^0V_8$ ○ 0 0 0	$^0V_9$ ○ 0 0 0	$^0V_{10}$ ○ 0 0 0	$^0V_{11}$ ○ 0 0 0	$^0V_{12}$ ○ 0 0 0	$^0V_{13}$ ○ 0 0 0	$^1V_{21}$ ○ B_1 in a decimal fraction. $\boxed{B_1}$	$^1V_{22}$ ○ B_3 in a decimal fraction. $\boxed{B_3}$	$^1V_{23}$ ○ B_5 in a decimal fraction. $\boxed{B_5}$	$^0V_{24}$... ○ 0 0 0 $\boxed{B_7}$
1	×	$^1V_2 \times {}^1V_3$	$^1V_4, {}^1V_5, {}^1V_6$	$\left\{\begin{matrix} ^1V_2 = {}^1V_2 \\ ^1V_3 = {}^1V_3 \end{matrix}\right\}$	$= 2n$	...	2	n	$2n$	$2n$	$2n$											
2	−	$^1V_4 - {}^1V_1$	$^2V_4$	$\left\{\begin{matrix} ^1V_4 = {}^2V_4 \\ ^1V_1 = {}^1V_1 \end{matrix}\right\}$	$= 2n - 1$	1	...	...	$2n-1$													
3	+	$^1V_5 + {}^1V_1$	$^2V_5$	$\left\{\begin{matrix} ^1V_5 = {}^2V_5 \\ ^1V_1 = {}^1V_1 \end{matrix}\right\}$	$= 2n + 1$	1	...	...	...	$2n+1$												
4	÷	$^2V_5 \div {}^2V_4$	$^1V_{11}$	$\left\{\begin{matrix} ^2V_5 = {}^0V_5 \\ ^2V_4 = {}^0V_4 \end{matrix}\right\}$	$= \frac{2n-1}{2n+1}$	...	...	...	0	0	...	...	...	...	...	$\frac{2n-1}{2n+1}$						
5	÷	$^1V_{11} \div {}^1V_2$	$^2V_{11}$	$\left\{\begin{matrix} ^1V_{11} = {}^2V_{11} \\ ^1V_2 = {}^1V_2 \end{matrix}\right\}$	$= \frac{1}{2} \cdot \frac{2n-1}{2n+1}$	...	2	...	...	...	...	...	...	...	...	$\frac{1}{2} \cdot \frac{2n-1}{2n+1}$						
6	−	$^0V_{13} - {}^2V_{11}$	$^1V_{13}$	$\left\{\begin{matrix} ^2V_{11} = {}^0V_{11} \\ ^0V_{13} = {}^1V_{13} \end{matrix}\right\}$	$= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} = A_0$	...	...	...	...	...	...	...	...	...	...	0	....	$-\frac{1}{2} \cdot \frac{2n-1}{2n+1} = A_0$				
7	−	$^1V_3 - {}^1V_1$	$^1V_{10}$	$\left\{\begin{matrix} ^1V_3 = {}^1V_3 \\ ^1V_1 = {}^1V_1 \end{matrix}\right\}$	$= n - 1\ (= 3)$	1	...	n	...	...	...	...	...	...	$n - 1$							
8	+	$^1V_2 + {}^0V_7$	$^1V_7$	$\left\{\begin{matrix} ^1V_2 = {}^1V_2 \\ ^0V_7 = {}^1V_7 \end{matrix}\right\}$	$= 2 + 0 = 2$	...	2	...	...	...	...	2										
9	÷	$^1V_6 \div {}^1V_7$	$^3V_{11}$	$\left\{\begin{matrix} ^1V_6 = {}^1V_6 \\ ^0V_{11} = {}^3V_{11} \end{matrix}\right\}$	$= \frac{2n}{2} = A_1$	...	...	...	...	...	$2n$	2	...	...	...	$\frac{2n}{2} = A_1$						
10	×	$^1V_{21} \times {}^3V_{11}$	$^1V_{12}$	$\left\{\begin{matrix} ^1V_{21} = {}^1V_{21} \\ ^3V_{11} = {}^3V_{11} \end{matrix}\right\}$	$= B_1 \cdot \frac{2n}{2} = B_1 A_1$	...	...	...	...	...	...	...	...	...	...	$\frac{2n}{2} = A_1$	$B_1 \cdot \frac{2n}{2} = B_1 A_1$	....	B_1			
11	+	$^1V_{12} + {}^1V_{13}$	$^2V_{13}$	$\left\{\begin{matrix} ^1V_{12} = {}^0V_{12} \\ ^1V_{13} = {}^2V_{13} \end{matrix}\right\}$	$= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2}$ ...	...	...	...	...	...	...	...	...	...	...	....	0	$\left\{-\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2}\right\}$				
12	−	$^1V_{10} - {}^1V_1$	$^2V_{10}$	$\left\{\begin{matrix} ^1V_{10} = {}^2V_{10} \\ ^1V_1 = {}^1V_1 \end{matrix}\right\}$	$= n - 2\ (= 2)$	1	...	...	...	...	...	...	...	...	$n - 2$							
13	−	$^1V_6 - {}^1V_1$	$^2V_6$	$\left\{\begin{matrix} ^1V_6 = {}^2V_6 \\ ^1V_1 = {}^1V_1 \end{matrix}\right\}$	$= 2n - 1$	1	...	...	...	...	$2n-1$											
14	+	$^1V_1 + {}^1V_7$	$^2V_7$	$\left\{\begin{matrix} ^1V_1 = {}^1V_1 \\ ^1V_7 = {}^2V_7 \end{matrix}\right\}$	$= 2 + 1 = 3$	1	...	...	...	...	...	3										
15	÷	$^2V_6 \div {}^2V_7$	$^1V_8$	$\left\{\begin{matrix} ^2V_6 = {}^2V_6 \\ ^2V_7 = {}^2V_7 \end{matrix}\right\}$	$= \frac{2n-1}{3}$	...	...	...	...	...	$2n-1$	3	$\frac{2n-1}{3}$									
16	×	$^1V_8 \times {}^3V_{11}$	$^4V_{11}$	$\left\{\begin{matrix} ^1V_8 = {}^0V_8 \\ ^3V_{11} = {}^4V_{11} \end{matrix}\right\}$	$= \frac{2n}{2} \cdot \frac{2n-1}{3}$	...	...	...	...	...	...	...	0	...	...	$\frac{2n}{2} \cdot \frac{2n-1}{3}$						
17	−	$^2V_6 - {}^1V_1$	$^3V_6$	$\left\{\begin{matrix} ^2V_6 = {}^3V_6 \\ ^1V_1 = {}^1V_1 \end{matrix}\right\}$	$= 2n - 2$	1	...	...	...	...	$2n-2$											
18	+	$^1V_1 + {}^2V_7$	$^3V_7$	$\left\{\begin{matrix} ^2V_7 = {}^3V_7 \\ ^1V_1 = {}^1V_1 \end{matrix}\right\}$	$= 3 + 1 = 4$	1	...	...	...	...	...	4										
19	÷	$^3V_6 \div {}^3V_7$	$^1V_9$	$\left\{\begin{matrix} ^3V_6 = {}^3V_6 \\ ^3V_7 = {}^3V_7 \end{matrix}\right\}$	$= \frac{2n-2}{4}$	...	...	...	...	...	$2n-2$	4	...	$\frac{2n-2}{4}$	...	$\left\{\frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{3} = A_3\right\}$						
20	×	$^1V_9 \times {}^4V_{11}$	$^5V_{11}$	$\left\{\begin{matrix} ^1V_9 = {}^0V_9 \\ ^4V_{11} = {}^5V_{11} \end{matrix}\right\}$	$= \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{4} = A_3$	...	...	...	...	...	...	...	...	0								
21	×	$^1V_{22} \times {}^5V_{11}$	$^0V_{12}$	$\left\{\begin{matrix} ^1V_{22} = {}^1V_{22} \\ ^0V_{12} = {}^2V_{12} \end{matrix}\right\}$	$= B_3 \cdot \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{3} = B_3 A_3$	...	...	...	...	...	...	...	...	...	...	0	$B_3 A_3$	....	......	B_3		
22	+	$^2V_{12} + {}^2V_{13}$	$^3V_{13}$	$\left\{\begin{matrix} ^2V_{12} = {}^0V_{12} \\ ^2V_{13} = {}^3V_{13} \end{matrix}\right\}$	$= A_0 + B_1 A_1 + B_3 A_3$	...	...	...	...	...	...	...	...	...	...	....	0	$\left\{A_3 + B_1 A_1 + B_3 A_3\right\}$				
23	−	$^2V_{10} - {}^1V_1$	$^3V_{10}$	$\left\{\begin{matrix} ^2V_{10} = {}^3V_{10} \\ ^1V_1 = {}^1V_1 \end{matrix}\right\}$	$= n - 3\ (= 1)$	1	...	...	...	...	...	...	...	...	$n - 3$							
					Here follows a repetition of Operations thirteen to twenty-three.																	
24	+	$^4V_{13} + {}^0V_{24}$	$^1V_{24}$	$\left\{\begin{matrix} ^4V_{13} = {}^0V_{13} \\ ^0V_{24} = {}^1V_{24} \end{matrix}\right\}$	$= B_7$	...	...	...	...	...	...	...	...	...	...	....	....	....	......	......	......	B_7
25	+	$^1V_1 + {}^1V_3$	$^1V_3$	$\left\{\begin{matrix} ^1V_1 = {}^1V_1 \\ ^1V_3 = {}^1V_3 \\ ^5V_6 = {}^0V_6 \\ ^5V_7 = {}^0V_7 \end{matrix}\right\}$	$= n + 1 = 4 + 1 = 5$ .... by a Variable-card. by a Variable card.	1	...	$n+1$	...	...	0	0										

Figure 2. Lovelace's program for calculating Bernoulli numbers. She uses the same format as Menabrea, but with many columns of explanation for her readers. Note that in line 1, she assigns the value 2n to multiple variables, and after line 23 she indicates a loop with the text, "Here follows a repetition of Operations thirteen to twenty-three."

$$mx + ny = d$$

$$m'x + n'y = d'$$

or

$$x = \frac{dn' - d'n}{n'm – nm'}$$

$$y = \frac{md' - m'd}{mn' - m'n}$$

Menabrea's instructions demonstrate the power of storing values in variables and reusing them. (See row 11, where he uses the same stored values to calculate y as he used for x.) Figure 2 shows Lovelace's program for calculating Bernoulli numbers. Note that in line 1, she assigns the value 2n to multiple variables, and after line 23 she indicates a loop with the text, "Here follows a repetition of Operations thirteen to twenty-three."

Winston Churchill said that, "Plans are of little importance, but planning is essential." In the case of Babbage's Analytical Engine, we see the truth of Churchill's words. Although the planned Analytical Engine was never built, the activity of creating algorithms for it allowed Babbage, Menabrea, and Lovelace the opportunity to explore ideas relevant to bridging the gap between human language and machine language. The next attempt to do so would come a century later.

PLANKALKÜL

In 1938, Konrad Zuse completed the computer he was building in the living room of his parents' apartment in Berlin. [2][3][4]. He had quit his job in airplane construction two years prior in order to work on his machine, which he dubbed the VersuchsModell 1 (Experimental Model 1) [5] or V1. After World War II, it was renamed Z1, so as not

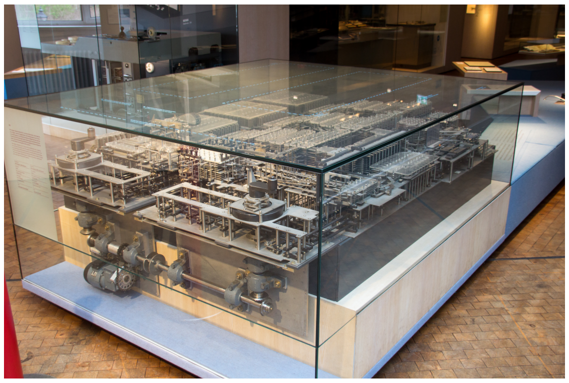
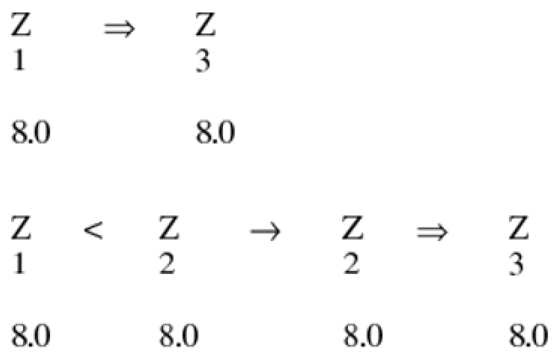


Figure 3. Replica of the Z1 in the German Museum of Technology in Berlin. [8] “There is a replica of this Model in the Museum of Traffic and Technology in Berlin. Back then it didn’t function well, and in that regard the replica is very reliable – it also doesn’t work well.” —Konrad Zuse [2][9]

to be confused with the German V-1 flying bomb. [5] Like the proposed Analytical Engine, the Z1 was mechanical. Zuse’s machine, however, used Boolean logic and binary floating-point numbers, the first freely programmable computer to do so. It did not hold a program in memory, but executed as it read from a punched tape. A separate tape reader provided input to the machine. [2] The Z1 and its construction plans were destroyed by Allied bombing in December of 1943, along with its successors, the Z2 [6] and Z3 [7]. The Z4 was transported in February 1945 from Berlin to Göttingen to prevent it from falling into the hands of the Soviets. As the war ended, Zuse again moved the Z4 to a stable in the Alpine village of Hinterstein. It was later acquired and refurbished by the Federal Polytechnic Institute (ETH) in Zurich. [5]

Zuse was self-taught with respect to computers, and he designed his own diagram and notation system to describe logical circuits, only discovering the existing propositional calculus in 1938. In 1939 he wrote of that discovery and added, “Now I plan creation of ‘Calculus of plans.’ There are series of concepts needed to clarify for this.” [10][11] The “Calculus of plans” Zuse developed was the Plankalkül, the first high-level programming language designed for a computer. [10]

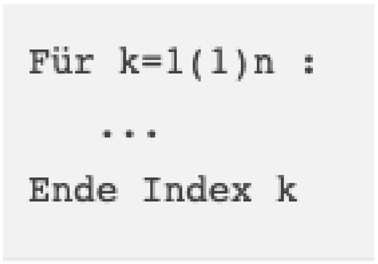


```
Z    ⇒    Z
1         3

8.0       8.0

Z  <  Z    →    Z    ⇒    Z
1     2         2         3

8.0   8.0       8.0       8.0
```

Figure 4. Plankalkül code was written with variables in a vertical format. This code calculates the max of two eight-bit integers, Z1 and Z2, and stores it in Z3. The guarded command (a single arrow) acts as an IF-THEN. The double arrow serves as an assignment operator. Variables are indicated as vertical lists of attributes in a prescribed order from top to bottom. The first attribute is the kind of variable. Each variable must be V (read-only), Z (read or write), or R (write only). The variables do not have names; to distinguish a variable of a given kind from others of that same kind, they are each given an index, and this index is the second vertical element. Since every variable is a combination of bits, the third element is the component index, indicating which bit is being referred to. This position might be left empty if the whole array of bits is being referred to, as in this example, where an array of 8 bits is used to express an integer number. The fourth vertical element indicates how many bits are included in the variable, its type. [12]

The Plankalkül was fairly sophisticated, considering that Zuse never actually implemented it. He did write algorithms in it, and obviously recognized the need for structural components such as loops, conditionals, and subroutines. It included a WHILE loop, which had a special form that functioned as a FOR loop, and “guarded commands” that function like an IF-THEN. Programs were numbered functions, which could be invoked with call-by-value variables. Variables could be of type bit, array of bits, or tuple. Types did not need to be declared, because each reference to the variable included its type. [12]

In Zuse’s syntax for Plankalkül, variables are written in a vertical format (See Figure 4), which



```
Für k=1(1)n :
    ...
Ende Index k
```

Figure 5. For loop structure in Superplan [14]

makes it difficult to reproduce in printed text. Since Zuse never implemented Plankalkül, he did not have to resolve how he would input the code. When Plankalkül was first implemented in 1975, a horizontal format was used for variable names. [10] Like the language of the Analytical Engine, Plankalkül served as a framework for thinking about what a programming language needs for functionality and for practical use by human beings.

SUPERPLAN

When Zuse's Z4 was acquired by the ETH in Zurich, one of its early users was Heinz Rutishauser. Rutishauser and his colleague, Ambros Speiser, were employed to study the mathematical implications of computers at the newly formed Institute for Applied Mathematics at ETH. They were assigned to learn the state of the art in computing and establish a computing program. They spent all of 1949 visiting first Howard Aiken at Harvard and then John von Neumann at Princeton, as well as other installations. However, when they returned in December of that year, they discovered that ETH had obtained Konrad Zuse's Z4. [13] Influenced by Plankalkül, Rutishauser developed Superplan, which introduced the key word "for" ("für" in German) for a loop over an array with an iterator included in the definition of the loop.

ENIAC CABLES

On the other side of the Atlantic, the Moore School of Electrical Engineering, a part of the University of Pennsylvania, was grappling with the difficulty of calculating artillery firing tables for large numbers of guns that the U.S. Army was developing for the war effort. Lieutenant Herman H. Goldstine, the liaison between the United States Army and Moore School, stated, "Such a trajectory involved about 750 multiplications and would take a human at least seven hours." [16] John Mauchly, the head (and only staff member) of the physics department at Ursinus College, came to the Moore School for the Defense Training Course for Electronics and stayed to accept a teaching position. His lab instructor during the course was J. Presper Eckert [17], who convinced him that vacuum tubes, which were notoriously unreliable, could be employed with good engineering practices to build a purely electronic computer. Mauchly wrote a memo proposing it, "The Use of High-Speed Vacuum Tube Devices For Calculating," in which he argued that "the result of one calculation, such as a single multiplication, is immediately available for further operation in any way which is dictated by the equations governing the problem, and these numbers can be transferred from one component to another as required, without the necessity of copying them manually onto paper or from one component to another, as is the case when step by step solutions are performed with ordinary calculating machines." [18]

Mauchly's memo was brought to Goldstine's attention, and Goldstine invited him to write a formal proposal, resulting in a contract, in April 1943, for the Moore School to construct the Electronic Numerical Integrator and Computer (ENIAC). Capable of adding 5,000 numbers in a second, the ENIAC was a technological wonder, allowing for the solution of problems that were previously unsolvable. Before its construction was finished, however, Mauchly and Eckert were already planning its successor, the EDVAC, which would be capable of storing programs. It happened that Goldstine met John von Neumann by chance on a railway platform in the summer of 1944 and began to tell him about the ENIAC project. Apparently, von Neumann's interest was piqued, causing Goldstine to report that "the whole atmosphere of our conversation changed from one of relaxed good humour to one more like the oral examination for the doctor's degree in mathematics." [19] Von Neumann joined the team and described the EDVAC in 1945 in a paper entitled "First Draft of a Report on the EDVAC." (Eckert and Mauchly left the Moore

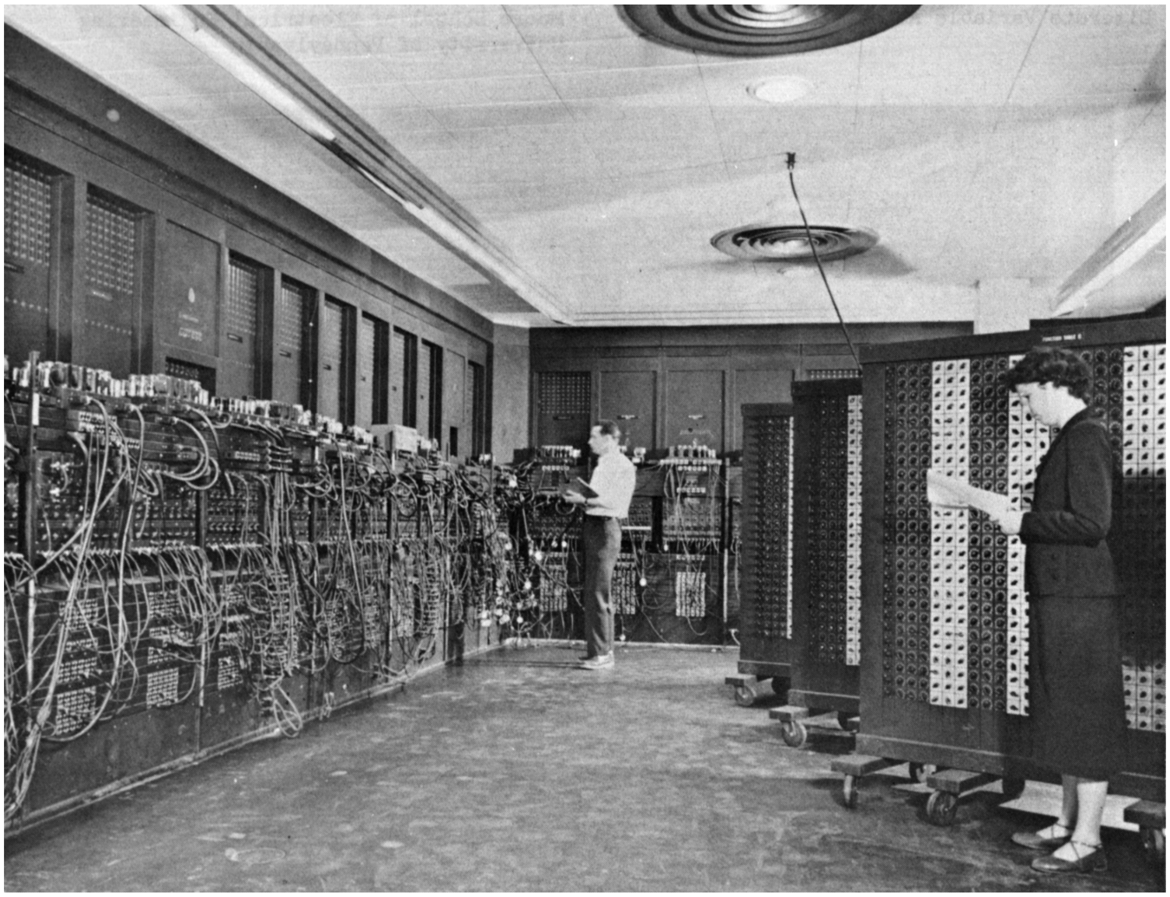


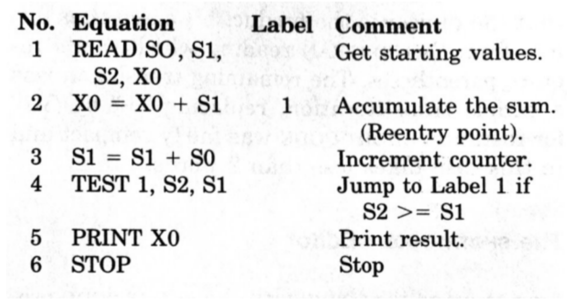
Figure 6. ENIAC (Electronic Numerical Integrator and Computer) in Philadelphia, Pennsylvania. [15] Glen Beck (background) and Betty Snyder (foreground). The ENIAC was programmed by moving its cables.

School around this time, due at least in part to a change in the school's policy regarding intellectual property, and the paper was distributed without referencing them. [20])

Somewhat ironically, the programming of computers was initially considered merely a task to be done, rather than an academic pursuit, and the work of programming the ENIAC was given to six women, the ENIAC six. They had no programming language, nor even an instruction manual. They programmed the ENIAC by moving its cables around, "often with only the circuit schematics to go by." [21] Thus, the programming language for the ENIAC was, essentially, cables.

MARK 1

Meanwhile, another woman was part of the programming team for the Mark I, the only other computer in existence. [21] Dr. Grace Hopper had received her PhD from Yale in 1934 and taught mathematics at Vassar [22] until she took a leave of absence to join the Navy Reserves in December of 1943, assuming she would be assigned to a team focused on code breaking. [21] Instead, she was assigned to the Bureau of Ordnance Computation Project at Harvard [22], where she began programming the Mark I, an electromechanical computer programmed with punch cards. [21] Though she worked strictly in machine language, Hopper was already beginning to formulate ideas about efficient use of subroutines. In her own words: "Any problem we solved, we found we did not need complete generality; we always knew something about what we were doing – that was what the problem was. And the answer was – we started writing subroutines, only we thought they were pieces of coding. And if I needed a



No.	Equation	Label	Comment
1	READ SO, S1, S2, X0		Get starting values.
2	X0 = X0 + S1	1	Accumulate the sum. (Reentry point).
3	S1 = S1 + S0		Increment counter.
4	TEST 1, S2, S1		Jump to Label 1 if S2 >= S1
5	PRINT X0		Print result.
6	STOP		Stop

Figure 7. This Short Code example calculates the sum of all numbers from S0 to S2 with an increment of S1.[24]

sine subroutine, angle less than pi/4, I'd whistle at Dick and say, 'Can I have your sine subroutine?' and I'd copy it out of his notebook. We soon found that we needed just a generalized format of these if we were going to copy them." [22]

OCTAL on the BINAC

In 1947 Eckert and Mauchly formed the first computer company, the Eckert-Mauchly Computer Corporation (EMCC), where they continued their work on the EDVAC with the BINAC and then the UNIVAC, the first large-scale commercially available computer. [20] Hopper, who was mustered out of the Navy after the war due to her age (40), [23] joined EMCC in 1949. She describes her first experience there: "They were building BINAC, a binary computer. We programmed it in octal. Thinking I was still a mathematician, I taught myself to add, subtract, and multiply, and even divide in octal. I was really good, until the end of the month, and then my checkbook did not balance! It stayed out of balance for three months until I got hold of my brother who's a banker. After several evenings of work he informed me that at intervals I had subtracted in octal. And I faced the major problem of living in two different worlds. That may have been one of the things that sent me to get rid of octal as far as possible." [22]

UNIVAC SHORT CODE

Mauchly quickly realized that if customers were going to buy computers, they would need software (though the word "software" was not yet in use), and he designed "Brief Code", which came to be called Univac Short Code, for the BINAC, making it the first programming language actually used on a computer. [20] Short Code was an interpreted language, which ran about 50 times slower than machine code. [25] It allowed for branching and calls to a library of functions. It was implemented by a summer intern, William Schmitt, who reported to Grace Hopper. Of Short Code, Hopper wrote, "I think this was the first thing that clued me to the fact that you could use some kind of a code other than the actual machine code…To us it was a pseudo-code." [22]

Betty Holberton of the ENIAC six was another EMCC employee. [26] In 1951, Holberton wrote a "Sort-Merge Generator" for the Univac 1. Its output was machine code, and, perhaps due to her musings regarding reusable code, Hopper was quick to recognize the significance of it. "You fed it the specifications of the files you were operating on, and the Sort-Merge Generator produced the program to do all of the input and output in the various tape units…It meant that I could do these things automatically; that *you could make a computer write a program*." (emphasis added) [22] In spite of frequent objections from the "Establishment" that a computer could not write a program, Hopper insisted that "I could make a computer do anything which I could completely define," and witnessed the creation of "a whole family of other generators." So far, though, she had not convinced anyone else of their import. [22]

The A-0 COMPILER

In October, 1951, Hopper received an assignment to build a set of mathematical subroutines for the UNIVAC 1, standardized for general use. "As I went along in the process of standardizing the subroutines, I recognized something else was happening in the programming group. We were using subroutines. We were copying routines from one program into another. There were two things wrong with that technique: one was that the subroutines were all started at line 0 and went on sequentially from there. When you copied them into another program, you therefore had to add to all those addresses as you copied

Figure 8. Grace Hopper [27]

them into the new program – you had to add to all those addresses. And programmers are lousy adders! The second thing that inhibited this was that programmers are lousy copyists! And it was amazing how many times a 4 would turn into a delta which was our space symbol, or into an A - and even Bs turned into 13s. All sorts of interesting things happened when programmers tried to copy subroutines. And there of course stood a gadget whose whole purpose was to copy things accurately and do addition. And it therefore seemed sensible, instead of having the programmers copy the subroutines, to have the computer copy the subroutines. Out of that came the A-0 compiler." [22]

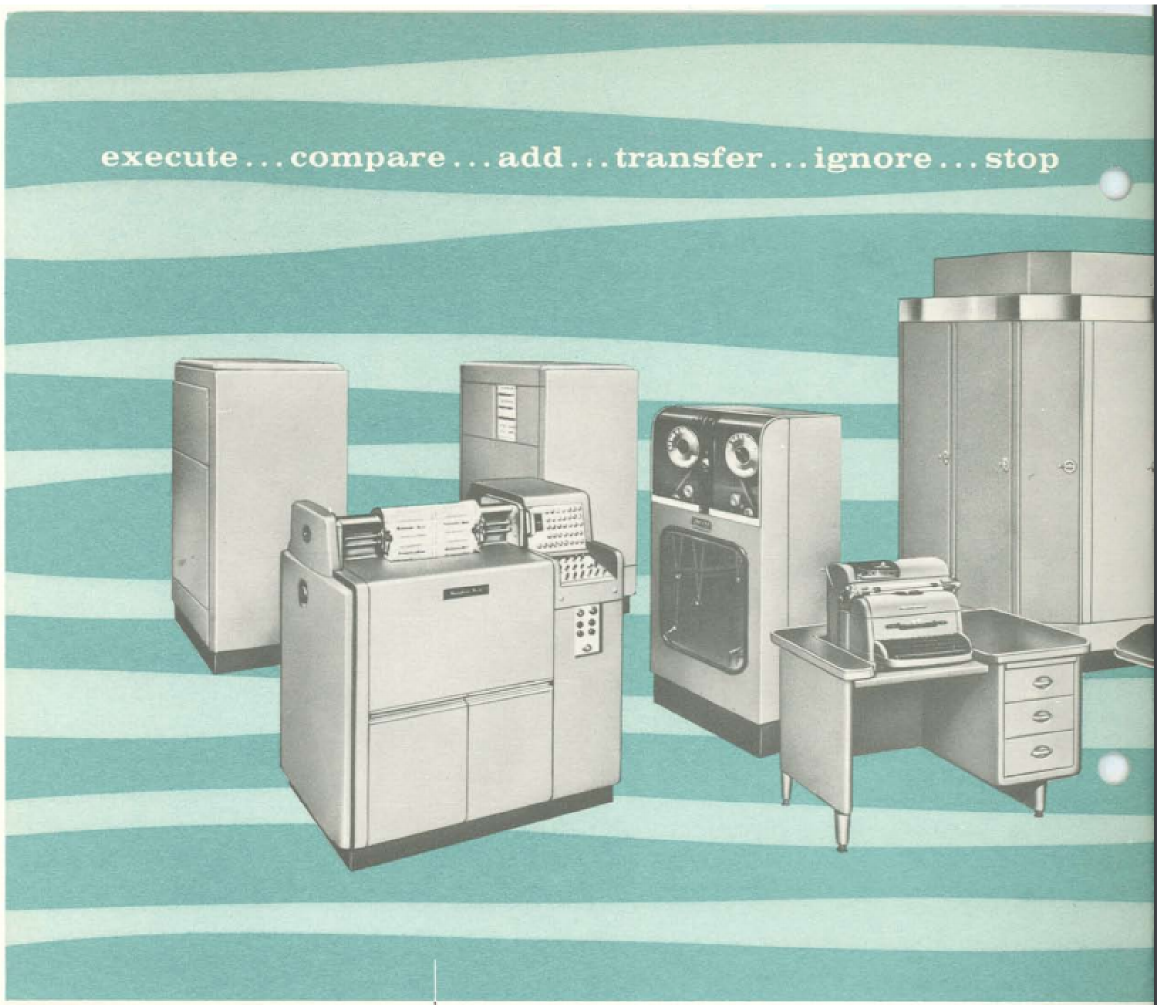
Though Hopper's account is worth reading for both enlightenment and entertainment, a summary must suffice here to explain a few of the significant elements of her A-0 compiler, the first of its kind. Emphasis was on allowing programmers to create executable programs quickly, rather than optimizing the machine code that was produced. In addition, it was important for the machine to be accessible to more and more people, including some who did not want to "learn octal code and manipulate bits." It did not occur to Hopper to make two passes in order to link with subroutines that had not been encountered yet. Instead, she created a "neutral corner" of memory in which she kept track of places where forward references were needed, a concept she says was inspired by making forward passes in basketball, which she played as an undergraduate at college. She documented all of her work on the compiler extensively, knowing that nobody would believe it was going to work. The idea of a machine writing machine code was still too outrageous for other members of the computing community to wrap their minds around it. [22] For several years in the mid 1950's, Hopper worked tirelessly to convince the computing community of the value of programming languages, compilers, subroutines, and a focus on software development in general. [23]

Justice requires an acknowledgement, at this point, of the independent efforts of Alick Glennie, who developed a programming language and compiler, known simply as "Autocode," for the Mark 1 computer at the University of Manchester in 1952. This is considered by some to be the first compiler. [28] There is much less information available about Autocode or Glennie than Hopper and her work. This is evidence, I believe, that Hopper's vision of software as a discipline and her persevering effort to convey this vision to the world at large was an even greater contribution than actually writing a compiler.

The compiler, though, was of enormous impact. The A-0, which focused on mathematical programming, was succeeded by the A-2, A-3, and AT-3 compilers. These last were eventually renamed ARITH-MATIC and MATH-MATIC by the marketing department of Remington Rand UNIVAC. [23][22]

The B-0 COMPILER (UNIVAC FLOW-MATIC)

In the meantime, Hopper had been thinking about a compiler for data processing. She had


execute . . . compare . . . add . . . transfer . . . ignore . . . stop

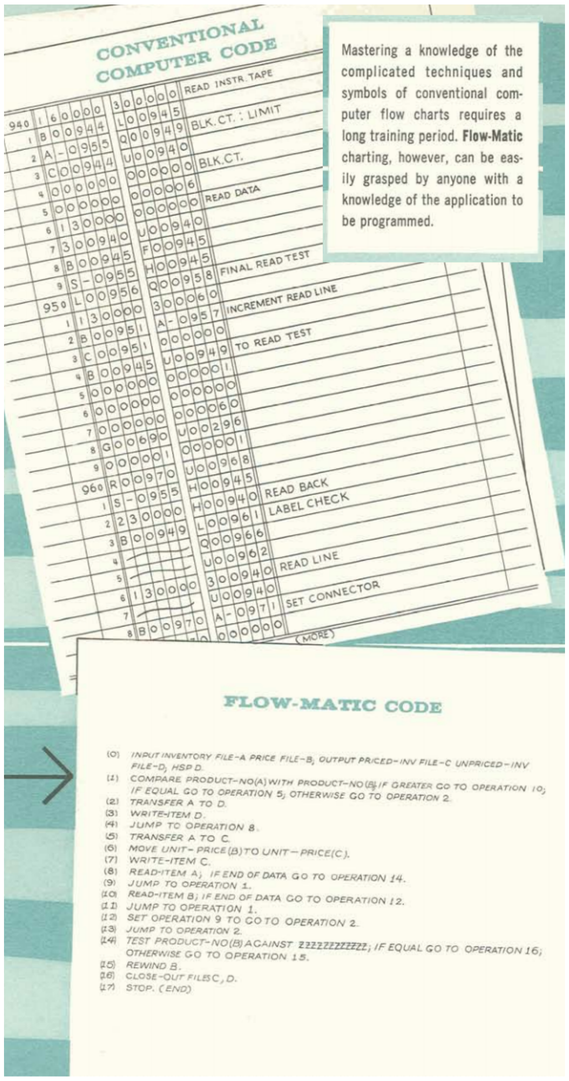
Features of UNIVAC®

FLOW-MATIC

Takes You Directly from Flow Chart to Finished Program

UNIVAC FLOW-MATIC is the most far-reaching development ever offered for automatic computer programming. It provides, for the first time, a means whereby the flow chart of the systems expert can be translated automatically, at electronic speed, into the language of the UNIVAC II Data-Automation System.

To program a new application, the user merely describes his systems flow chart in the English-language instructions of FLOW-MATIC. These act as a signalling index to the computer routines of the FLOW-MATIC library. When read by the UNIVAC system, the instructions cause the computer to generate for itself the various subroutines required to process the problem. It then assembles these subroutines into a finished program and records the program on magnetic tape.



CONVENTIONAL COMPUTER CODE

		300000	READ INSTR. TAPE
940	160000	L00945	
1	B00944	Q00949	BLK. CT. : LIMIT
2	A-0955	U00940	
3	C00944	000000	BLK.CT.
4	000000	000006	
5	000000	000000	READ DATA
6	130000	U00940	
7	300940	F00945	
8	B00945	H00945	
9	S-0955	Q00958	FINAL READ TEST
950	L00956	300060	
1	130000	A-0957	INCREMENT READ LINE
2	B00951	000000	
3	C00951	U00949	TO READ TEST
4	B00945	000001	
5	000000	000000	
6	000000	000060	
7	000000	U00296	
8	G00690	000001	
9	000001	U00968	
960	R00970	H00945	
1	S-0955	H00940	READ BACK
2	230000	L00961	LABEL CHECK
3	B00949	Q00966	
4		U00962	
5		300940	READ LINE
6	130000	U00940	
7		A-0971	SET CONNECTOR
8	B00970	000000	

(MORE)

Mastering a knowledge of the complicated techniques and symbols of conventional computer flow charts requires a long training period. **Flow-Matic** charting, however, can be easily grasped by anyone with a knowledge of the application to be programmed.

FLOW-MATIC CODE

```
(0)  INPUT INVENTORY FILE-A PRICE FILE-B; OUTPUT PRICED-INV FILE-C UNPRICED-INV
     FILE-D; HSP D.
(1)  COMPARE PRODUCT-NO(A) WITH PRODUCT-NO(B); IF GREATER GO TO OPERATION 10;
     IF EQUAL GO TO OPERATION 5; OTHERWISE GO TO OPERATION 2.
(2)  TRANSFER A TO D.
(3)  WRITE-ITEM D.
(4)  JUMP TO OPERATION 8.
(5)  TRANSFER A TO C.
(6)  MOVE UNIT-PRICE(B) TO UNIT-PRICE(C).
(7)  WRITE-ITEM C.
(8)  READ-ITEM A; IF END OF DATA GO TO OPERATION 14.
(9)  JUMP TO OPERATION 1.
(10) READ-ITEM B; IF END OF DATA GO TO OPERATION 12.
(11) JUMP TO OPERATION 1.
(12) SET OPERATION 9 TO GO TO OPERATION 2.
(13) JUMP TO OPERATION 2.
(14) TEST PRODUCT-NO(B) AGAINST ZZZZZZZZZZZZ; IF EQUAL GO TO OPERATION 16;
     OTHERWISE GO TO OPERATION 15.
(15) REWIND B.
(16) CLOSE-OUT FILES C, D.
(17) STOP. (END)
```

Figure 9. FLOW-MATIC Code [29]

found that the people who worked in data processing were "business trained," rather than mathematically trained; "they were word-manipulators rather than mathematics people." Another observation she made was that whereas mathematics and engineering had a well-defined language, there was no such language for data processing. "We took something over 500 data processing programs and sat down to write out what people were doing when they did data processing. We finally identified about 30 verbs which seemed to be the operators of data processing." [22]

In a report in 1953, Hopper proposed to management that her team write two separate compilers, a symbolic one for mathematics, the A-0, and a data processing compiler to translate English-based code. She was told she could not do that, because computers could not understand English words. Hopper and her team, however, understood the nature of symbolic representation, which could extend to English-based words as well as to mathematical symbols, and in January 1955 they submitted the formal proposal for the B-0 compiler for data processing. The language was to be variable-length English words separated by spaces. They built a pilot model to prove the concept, choosing English words in which the first and third letter combinations were unique among possible key words, in order to make parsing easy. On the back of the report to management, they printed an English-language based program (below) that would run in the new compiler. To further prove the concept, they made it work with French words or German words instead, but this so alarmed the management that the team dropped all references to foreign languages. B-0 was eventually accepted and renamed FLOW-MATIC to match the company's other compiler products.

This is the note to management and the example of English-based code that Hopper's team included on the back of their report on the B-0 compiler: "Dear Kind Management: If you come down to the machine room, we'll be delighted to run this program for you. INPUT INVENTORY FILE A; PRICE FILE B; OUTPUT PRICED INVENTORY FILE C. COMPARE PRODUCT #A WITH PRODUCT #B. IF GREATER, GO TO OPERATION 10; IF EQUAL, GO TO OPERATION 5; OTHERWISE GO TO OPERATION 2. TRANSFER A TO D; WRITE ITEM D; JUMP TO OPERATION 8; REWIND B; CLOSE OUT FILE C AND D; STOP"

COBOL

The development of COBOL (Common Business-Oriented Language) was undertaken by the Department of Defense, with a goal of creating a standard data-processing language that would not need to be re-written to run on a different computer. [30] A committee (the Short Range Committee, later renamed the COBOL committee) of nine members was tasked with defining the specifications of the language. Jean Sammet, one of the nine committee members, reports that, "eventually over 25 people participated in some phase of the basic COBOL language design; this large group included two people who worked for Grace, but Grace herself was not a member of the committee that defined COBOL. . . . Thus while her indirect influence was very important, regrettably the frequently repeated statements that 'Grace Hopper developed COBOL' . . . are just not correct. Grace's primary contribution to COBOL was indirect, and via FLOW-MATIC. It was the only business-oriented programming language in use at the time the COBOL development started (aside from AIMACO which was a dialect of FLOW-MATIC). Without that existing practical use of FLOW-MATIC I doubt that we would have had the courage to develop a language such as COBOL. (The other significant input to the early COBOL work was Commercial Translator which was a set of specifications from IBM but it had not yet been implemented.) Thus, in my view, without FLOW-MATIC we probably never would have had a COBOL. The practical experience of implementing and using that type of language was priceless." [23] The first version of COBOL was COBOL 60, and "the U.S. government declared that any computer manufacturer wishing to do business with the government had to offer COBOL, unless it could show it has a better-performing language." [31] Compilers for the new language were created by hardware manufacturers.

SPEEDCODING AND FORTRAN

Around the same time that Hopper proposed her two compilers (1953), John Backus, a mathematician working for IBM, was performing a similar task. Backus had created Speedcoding, the first high-level programming language for an IBM, earlier that year, to support computation with floating point numbers. It was an interpreted language, which substantially reduced the time it took to create programs, but took more than ten times as long to run them. The program itself took up about 30% of the memory available on the IBM 701 for which it was designed. Late in 1953, Backus proposed that IBM develop a practical alternative to assembly language for the IBM 704 mainframe. [33] Backus's boss quickly agreed [22] and his team had created a draft specification for The IBM Mathematical Formula Translating System by November of the following year. The first FORTRAN compiler was delivered in 1957. [33] Unlike Speedcoding, FORTRAN was designed to generate code with "performance approaching that of hand-coded assembly language." [33] Of this need for efficiency, Backus said, "Most of the early systems we've seen, and many others, had hidden a lot of gross inefficiencies by virtue of the fact that most computing time was being spent in floating point subroutines. Because of that you could get away with a lot of clumsy treatment of looping and indexing, and references to arrays that escaped unnoticed because of all the time that was chewed up by floating point subroutines. But now we were confronted with the 704 which, for the first time, had built-in floating point and indexing, and we knew that this would make our goal of producing efficient object code very difficult because there was nowhere to hide inefficiencies. You couldn't do clumsy calculations of subscript combinations and get away with it in that machine." [22]

FORTRAN was an instant hit with scientists, who appreciated its ease of use, efficiency, and transportability across hardware platforms. FOR-

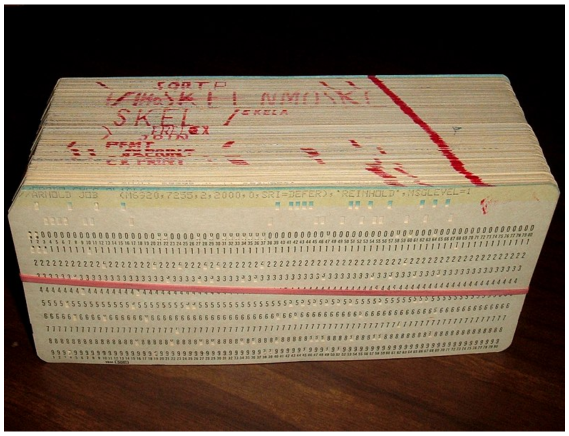


Figure 10. A deck of punched cards comprising a computer program. The deck was created circa 1969. Individual subroutines are marked in red on the sides of the cards, and the markings show the effects of editing, as cards are added, replaced or reordered.[32]

TRAN II was released the following year (1958), introducing subprograms with shared data space. FORTRAN IV, released in 1962, contained type statements, the logical-if statement, and procedure names passed as arguments. [33]

ALGOL

Not content with the definition of two programming languages, Backus traveled to the ETH in Zurich for an international conference, also attended by Rutishauer, to discuss a universal algorithmic language, which they dubbed ALGOL. ALGOL was an abstract representation for defining the standards of a language, but was also implemented specifically as ALGOL-58, and then ALGOL 60, [22] which became the standard language for the publication of algorithms.

CONCLUSION

The formative period of computer programming languages lasted until the late 1950's. Emergent ideas during this time period included interpreted languages, compiled languages, spoken-language based programming languages, platform independence, efficiency of programmer time, efficiency of machine code, and foundational structural elements of a programming language, such as variable types, while loops, for loops, conditional statements, branching of control flow, and generalized subroutines.

The Development of COBOL, FORTRAN, and ALGOL marks an important paradigm shift in the design of programming languages and the development of software. Previous work had been done by individuals and small teams, possessing the vision of programming languages and persevering in an environment of skepticism and outright disbelief, creating languages geared toward specific hardware. By 1960, however, the concept of high-level programming languages had gained general acceptance, English-language symbols had been proven feasible, and the computing community was beginning to establish standards for programming languages. The following two decades saw an explosion of new languages and paradigms. Most of these inherited directly from one or more of these three monumental achievements. All of them grew out of the vision, creativity, and effort of the mathematicians who dared to believe that human beings could speak to machines in our own language.

Further research on this topic would be helpful to understand the explosion of languages and paradigms during the 1970's and 1980's, and the further development and refinement of languages and concepts through to the present. A focus on the hardware technologies that accompanied each step in the evolution of programming languages would also be illuminating.

ACKNOWLEDGMENT

Thanks to Gavin Jensen (no relationship to the author) for reviewing this paper.

■ REFERENCES

1. Stephen Wolfram, "Untangling the Tale of Ada Lovelace," 2015.
2. Wikipedia contributors, "Z1 (computer)," 2020.
3. Friedrich L. Bauer, *Origins and Foundations of Computing: In Cooperation with Heinz Nixdorf Museums Forum.* Springer Science Business Media., 2009.
4. Konrad Zuse, *Plankalkül.* Gesellschaft für Mathematik und Datenverarbeitung, 1976.
5. Martin Campbell-Kelly, "Obituary: Konrad Zuse,"
6. Wikipedia contributors, "Z2 (computer)," 2020.
7. Wikipedia contributors, "Z3 (computer)," 2020.
8. Mike Peel, "Replica of the Z1 in the German Museum of Technology in Berlin; www.mikepeel.net."
9. Hans Dieter Hellige, *Geschichten der Informatik - Visionen, Paradigmen, Leitmotive.* Springer-Verlag Berlin Heidelberg, 2004.
10. Wikipedia contributors, "Plankalkül," 2020.
11. Raúl Rojas and Ulf Hashagen, *The First Computers: History and Architectures.* MIT Press, 2002.
12. Raúl Rojas and Cüneyt Göktekin and Gerald Friedland and Mike Krüger and Olaf Langmack and Denis Kunib, "Plankalkül: The First High-Level Programming Language and its Implementation," tech. rep., Takustr.9,14195 Berlin, Germany, 2000.
13. J J O'Connor and E F Robertson, "Heinz Rutishauser Biography," 2008.
14. Michael Wirth, "The Early Years of Coding," *Craftofcoding.wordpress.com*, vol. 05/05/2017.
15. U.S. Army, "ENIAC."
16. Stephen G. Nash, ed., *A History of Scientific Computing.* ACM Press, 1990.

17. John Mauchly, *John W. Mauchly papers*. University of Pennsylvania: Kislak Center for Special Collections, Rare Books and Manuscripts, ms. coll. 925 ed., 1908-1980.
18. John Mauchly, "The Use of High-Speed Vacuum Tube Devices for Calculating." 1942.
19. J J O'Connor and E F Robertson, "Herman Heine Goldstine," 2008.
20. Wikipedia contributors, "John Mauchly," 2020.
21. R. Candler, "Beyond Marie Curie: Grace Hopper and the ENIAC Six," *IEEE Potentials*, vol. 39, no. 3, May-June 2020.
22. Richard L. Wexelblat, ed., *History of Programming Languages*. Academic Press, A Subsidiary of Harcourt, Brace, Jovanovich, 1981.
23. Jean E. Sammet, "Farewell to Grace Hopper - end of an era!," *Communications of the ACM*, vol. 35, issue 4, 1992.
24. William F. Schmitt, "The Univac Short Code," *Annals of the History of Computing*, vol. Volume 10, Number 1, 1988.
25. Wikipedia contributors, "Short Code (computer language)," 2020.
26. Wikipedia contributors, "Betty Holberton," 2020.
27. U.S. Dept. of Defense, "Grace Hopper."
28. Wikipedia contributors, "Autocode," 2020.
29. *Introducing a New Language for Automatic Programming Univac Flow-Matic*. Sperry Rand Corporation. Univac Data Processing Division, 1957.
30. George Strawn and Candace Strawn, "Grace Hopper: Compilers and Cobol," *IT Professional*, vol. Volume: 17, Issue: 1, Jan.-Feb. 2015.
31. Jim Strothman, "Yesterday's Languages Built Today's Software," *Software Magazine*, vol. 9, Iss. 4, Mar 1989.
32. Arnold Reinhold, "https://commons.wikimedia.org/wiki/File:Punched_card_program_deck.agr.jpg."
33. Michael Metcalf, "The Seven Ages of Fortran," *JCST*, vol. 11 No. 1, April 2011.

Melanie Jensen is pursuing a Master's degree in Computer Science, with an emphasis on Human Computer Interaction. Melanie received a B.S. in Computer Science from BYU in 1993. She is the mother and homeschool teacher of four children. She loves efficient and effective processes, reading, good food, little kids, big piles of data, teaching people about homeschooling, word games, logic games, board games, teaching, learning, traveling, and dreaming of better ways to teach math, mostly involving computers. She is pursuing ways to use computers to teach math more effectively, to teach everything more effectively, to democratize education, and to participate in lifting people out of poverty via accessible, high-quality, free education.