**Machine Learning**

Machine learning is a field of study in artificial intelligence which focuses on the development of algorithms to learn from data and perform tasks without explicit programming. The core objective of machine learning is to generalize from its experiences in order to accurately perform tasks on new, unseen examples.

[Tom M. Mitchell](https://en.wikipedia.org/wiki/Tom_M._Mitchell) provided a widely quoted, more formal definition of the algorithms studied in the machine learning field: ” A computer program is said to learn from experience E with respect to some task T and some performance measure P if its’ performance on T, as measured by P, improves with experience E.”

As an example suppose that you have an email program that watches which meals you do or do not value spam, and based on that learns how to better filter spam. What is the task T in this setting? In this case, T would be the task of classifying emails as spam or not spam. The experience E would be the act of the program watching you labels spam emails as spam or not spam. The performance measure P would be the numbers of emails correctly marked as spam or not spam.

Machine learning approaches are traditionally divided into three broad categories, which correspond to learning paradigms, depending on the nature of the "signal" or "feedback" available to the learning system.

There are several different machine learning algorithms, but the main two types are called supervised and unsupervised learning algorithms.

In supervised learning, the machine is taught to perform a task. The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs.

In unsupervised learning, the model is left to learn by itself. Unlabeled input data is given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).

Both reinforcement learning and recommender systems are other types of machine learning that will be talked about later, however, the two most used types of algorithms are supervised and supervised learning

Another thing that this course will focus on is providing practical advice for applying machine learning algorithms effectively. Teaching about learning algorithms Is like giving a set of tools, and equally important is teaching how to apply these tools.

This course aims to develop students' skills in designing and building robust machine learning and AI systems. By applying machine learning best practices, practitioners can increase their chances of success and avoid wasted effort.

**Supervised Learning**

Supervised learning, formally, is a machine learning approach where input feature objects are paired with labeled desired output in order to train a model. When this training data is processed, the model builds a function that maps new data to expected output values based on parameters that were learned through training. Essentially, we are giving the algorithm a dataset where each data point has “right answers” given, and the task of the algorithm is to produce more of these right answers.

One popular use case for this algorithm is for a regression problem, where you are trying to predict a continuous valued output based on the given inputs. For example, suppose your friend is looking to sell their house and looking to see how much their house is worth. What they have is a dataset consisting of labeled pairs of the sizes of houses and its associated price. A supervised learning algorithm will learn from these pairs and will try to learn a mapping from the sizes of a house to the price of a house from the dataset. The model can then be used to predict the value of your friend’s house based on the size of their house.

Another use case of supervised learning is for classification problems, where you are trying to predict a discrete value output. For example, they give a breast cancer diagnosis problem where you are trying to classify whether a tumor is malignant or benign. Given the size of the tumor and the age of the patient labeled with true answers, the model will learn a mapping that separates between classes and determine whether or not a new patient has a malignant or benign tumor.

Machine learning algorithms can utilize multiple features or attributes to make predictions, and for some problems, you want an “infinite” number of features or cues in order to make more accurate predictions. One of the ways you can achieve this is through the support vector machine mechanism, however, it is important to keep in mind that this might not be the case for other problems. You should have a number of features that are descriptive of the object. The number of features should not be too large, because of the curse of dimensionality; but should contain enough information to accurately predict the output.

**Unsupervised Learning**

Unsupervised learning is a method in machine learning where, in contrast to supervised learning, algorithms learn patterns exclusively from unlabeled data. The machine, rather than learning the labels of each datapoint, clusters the data into distinct groups according to similar patterns in their input data. More precisely, the hope is that through mimicry, which is an important mode of learning in people, the machine is forced to build a concise representation of its world and then generate imaginative content from it. This type of learning is best suited for cases where you have large sets of data that are hard to label.

Neural network tasks are often categorized as discriminative (recognition) or generative. Often but not always, discriminative tasks use supervised methods and generative tasks use unsupervised; however, the separation is very hazy. For example, object recognition favors supervised learning but unsupervised learning can also cluster objects into groups.

There are several different use cases for unsupervised learning. Google News contains a vast amount of new stories everyday and groups them into cohesive news stories using clustering to categorize news stories based on content. Unsupervised learning aids in understanding genomics by identifying patterns in gene expression data. Other applications include organizing computer clusters, social network analysis, market segmentation, and astronomical data analysis.

As an example of how unsupervised learning works, we present the cocktail party algorithm, which separates audio sources from single recordings without prior knowledge of the individual voices. Take an example where there are two sources occurring at once during a recording The unsupervised learning algorithm should look at recording and should discriminate the audio into two distinct clusters, each one matched with one of the audio sources. From there, we can then extract the different clusters and now we have a way to extract overlapped audio voices from a recording.

While this algorithm may seem long and complicated, the cocktail party algorithm can be done in one line of code using a proper programming environment. For this reason, this course will be using Octave, which is a free open-source programming environment that allows for the building of machine learning algorithms to be done in just a few lines of code. It is for this reason that people will first prototype their models on programming such as Octave since they are fast and easy to build. Only after successfully building in Octave do developers migrate to other languages.

**ALU**

An ALU (Arithmetic and Logic Unit) is the "mathematical brain" of a computer that performs calculations and logical operations. The arithmetic unit handles all arithmetic and bitwise operations on integer binary numbers in the computer.

The ALU is made up of smaller circuits that make up the functionalities of the component. The simplest adding circuit is a "half adder" that adds two binary digits and produces a sum and carry bit. A "full adder" is a more complex circuit that adds three binary digits and handles the carry bit. An 8-bit ripple carry adder adds two 8-bit binary numbers by connecting half and full adders in a chain.

When two 8-bit numbers are added and there is a carry into the 9th bit, it means the sum of the two numbers is too large to fit into 8-bits. This is called an overflow. If we want to avoid overflows, we can extend our circuit with more full adders, allowing us to add 16 or 32 bit numbers, at the cost of speed. Modern computers use a slightly different adding circuit called a ‘carry-look-ahead’ adder which is faster, but ultimately does exactly the same thing-- adds binary numbers

The arithmetic unit of an ALU also performs other math operations, such as subtraction and incrementing, by combining logic gates. Some of the arithmetic operations are as follows:

* Add: A and B are summed and the sum appears at Y and carry-out.
* Add with carry: A, B and carry-in are summed and the sum appears at Y and carry-out.
* Subtract: B is subtracted from A (or vice versa) and the difference appears at Y and carry-out. For this function, carry-out is effectively a "borrow" indicator.
* Subtract with borrow: B is subtracted from A (or vice versa) with borrow (carry-in) and the difference appears at Y and carry-out (borrow out).
* Two's complement (negate): A (or B) is subtracted from zero and the difference appears at Y.
* Increment: A (or B) is increased by one and the resulting value appears at Y.
* Decrement: A (or B) is decreased by one and the resulting value appears at Y.
* Pass through: all bits of A (or B) appear unmodified at Y. This operation is typically used to determine the parity of the operand or whether it is zero or negative, or to load the operand into a processor register.

The logic unit of an ALU performs logical operations, such as AND, OR, and NOT, and tests numerical conditions. Some operations are as follows:

* AND: the bitwise AND of A and B appears at Y.
* OR: the bitwise OR of A and B appears at Y.
* Exclusive-OR: the bitwise XOR of A and B appears at Y.
* Ones' complement: all bits of A (or B) are inverted and appear at Y.

ALUs use flags to indicate specific statuses, such as zero, negative, and overflow, which are used to control program execution.

* Carry-out, which conveys the carry resulting from an addition operation, the borrow resulting from a subtraction operation, or the overflow bit resulting from a binary shift operation.
* Zero, which indicates all bits of Y are logic zero.
* Negative, which indicates the result of an arithmetic operation is negative.
* Overflow, which indicates the result of an arithmetic operation has exceeded the numeric range of Y.
* Parity, which indicates whether an even or odd number of bits in Y are logic one.

**Binary Numbers**

Computers are complex systems which require methods of representing values and letters. Computers do this by representing numeric value as a group of bits, such as bytes or words. The encoding of numerical value to bit patterns is chosen for convenience of the operation of the computer.

Single binary values represent numbers; 0 and 1 can be used to represent "true" and "false," allowing for information beyond these two values. Binary numbers work similarly to decimal numbers, using a base-two system with multipliers that are powers of two to represent larger numbers. For example, 73 is equal to seven 10's and three 1's in decimal, hence 73 in decimal form. For the binary representation, 73 is equal to one 64's, one 8's, and one 1's, hence 01001001 in binary form

In the past, 8-bit systems were very commonly used, where each binary digit (bit) represents a certain value, with a range of 0-255. In fact, 8-bits is such a common size in computing, it is known as a byte. As the amount of data gets larger, we require larger numerical representations leading to numbers using larger bit sizes. The amount of possible combinations of bits doubles with each added bit, leading to numbers of possible combinations to be 2^b where b is the number of bits in the representation. Along with a larger range of numbers, there also comes a need to represent negative values and decimal values. Here are some of the ways that computers do that:

* Representing negative numbers - Computers using 32 bit numbers represent negative numbers using the first bit, typically reserving 31 bits for the number itself.
* Representing more numbers - 64-bit numbers expand the range of representable values to around 9.2 quintillion.
* Representing non-whole numbers - Computers use floating-point numbers to represent non-whole numbers, with a significand and exponent.
  + Example: 625.9 = 0.6259 \* 10^3, where 0.6529 is the significand, and 3 is the exponent
  + In 32 bit floating point numbers, often referred to as a float, the first bit is used for sign, 8 bits are used for exponent, last 23 are for significand.
  + In 64 bit floating point numbers, often referred to as a double, the first bit is used for sign, 11 bits are used for exponent, last 52 are for significand.
  + The advantage of this scheme is that by using the exponent we can get a much wider range of numbers, even if the number of digits in the significand, or the "numeric precision", is much smaller than the range.

Aside from numbers, text is also represented in computers using encoding schemes like ASCII, which assigns binary numbers to letters. Using a 7-bit representation, we can encode 128 symbols that can be mapped to lowercase letters, uppercase letters, digits 0 - 9 and symbols such as @ and punctuation marks. ASCII also includes a selection of special command codes, corresponding to things like the new-line character.

With this encoding scheme, we are able to universally exchange information, however, only designed for the English alphabet. Other countries could have other language characters encoded by using 8-bit representations, though this caused problems when exchanging information across different languages. For this reason, Unicode was devised to provide a universal encoding scheme for characters from multiple languages, using a 16-bit representation to represent every single character in every language ever used.

**Boolean Logic**

Computers use binary (1s and 0s) to represent information, as it offers distinct signals and aligns with Boolean Algebra.

Today, all modern general-purpose computers perform their functions using two-value Boolean logic; that is, their electrical circuits are a physical manifestation of two-value Boolean logic. They achieve this in various ways: as voltages on wires in high-speed circuits and capacitive storage devices, as orientations of a magnetic domain in ferromagnetic storage devices, as holes in punched cards or paper tape, and so on

Of course, it is possible to code more than two symbols in any given medium. For example, one might use respectively 0, 1, 2, and 3 volts to code a four-symbol alphabet on a wire, or holes of different sizes in a punched card. In practice, the tight constraints of high speed, small size, and low power combine to make noise a major factor. This makes it hard to distinguish between symbols when there are several possible symbols that could occur at a single site. Rather than attempting to distinguish between four voltages on one wire, digital designers have settled on two voltages per wire, high and low.

Boolean Algebra, developed by George Boole, provides systematic rules for logical operations using true and false values. It differs from elementary algebra in two ways. First, the values of the variables are the truth values true and false, usually denoted 1 and 0, whereas in elementary algebra the values of the variables are numbers. Second, Boolean algebra uses logical operators such as conjunction (and) denoted as ∧, disjunction (or) denoted as ∨, and the negation (not) denoted as ¬

In terms of the computer architecture, the three fundamental Boolean operations, being NOT, AND, and OR, can be built using electrically controlled switches called transistors. NOT negates a single input, AND requires both inputs to be true for a true output, while OR requires only one input to be true.

The construction of these Boolean operations can be done by combining a series of logic gates. By combining more logic gates, more complex boolean operations can be created, such as XOR (exclusive OR), which is useful in computation. Computer engineers design processors using larger blocks, including logic gates and components built from gates in order to build larger abstractions.

Abstractions allow for complex systems to be built without considering the underlying implementation details, such as transistors or electrical signals. Computers evolved from electromechanical devices to electronic computers with transistors, which can represent "true" and "false" with binary states.

**CPU Design**

The instruction set contains expensive operations like divide, graphics, operations, encryption, etc. are implemented into the ALU hardware which does make design harder, but makes the system more capable. This extra circuitry makes the ALU bigger and more complicated to design, but also more capable - a complexity-for-speed tradeoff that has been made many times in computing history.

High clock speeds and fancy instructions set lead to another problem: getting data in and out of the CPU quickly off. In this case, the bottleneck is RAM as data between RAM and CPU has to travel through bus lines. One solution to this problem is by connecting a small piece of RAM onto the CPU, which we call the cache. By having a cache, the RAM can transmit not just one single value, but a whole block of data. Instead of going back and forth between RAM, we have a chunk of saved data that is much easier for the CPU to access. When the requested data from the RAM already exists on the cache, we call that a cache hit, whereas we call it a cache miss when the information we want to access is not on the cache. Cache can also be used as a scratch space for intermediary value when performing large computations. Generally speaking, more cache means more performance, due to reduced stalling.

With the implementation of the cache, we can also update the information much faster by changing the values in the cache rather than going back to the RAM, however, there now arises a different issue. Changes made to the data in cache are not replicated into RAM. The computer needs some way of knowing when the cache information is changed in order to change the information within RAM accordingly. The solution to this is a dirty bit. Each block of memory that the cache stores is given a flag known as the dirty bit which tells us whether we need to update the RAM data once we free the cache.

Another trick to boosting CPU performance is through instruction pipelining. Pipelining improves performance by allowing a number of instructions to work their way through the processor at the same time. We can utilize parallelization across the pipeline to improve throughput. Doing this allows us to better utilize the CPU to make sure we are doing as much work at once as possible.

With this method, then comes the issues of dependence between instructions. Pipelined processors have to look ahead for data dependencies, and if necessary, stall their pipelines to avoid problems. A technique to combat this is called out-of-order execution, in which we dynamically reorder instructions with dependencies in order to minimize stalls and keep the pipeline moving.

Another big hazard with pipelining are conditional jump operations, which change the execution flow of a program depending on a value. Rather than performing a long stall, high end processors perform what is called speculative execution, in which the processor guesses which way they are going to go, and start filling their pipeline with instructions based on that guess. If they make a wrong guess however, they have to flush the pipeline. To minimize these effects of flushes, branch prediction is done, where the hardware makes educated guesses on whether a particular branch will be taken

Outside of the techniques, there also comes the superscalar, multi-core and multi-processor systems, in which we increase the amount of cores, functional units like ALUs, or CPUs to increase the amount of instruction that can be run.

**CPU**

A central processing unit (CPU)—also called a central processor or main processor—is the most important processor in a given computer. Its electronic circuitry executes instructions of a computer program, such as arithmetic, logic, controlling, and input/output (I/O) operations. This role contrasts with that of external components, such as main memory and I/O circuitry,[1] and specialized coprocessors such as graphics processing units (GPUs).

A CPU consists of a RAM module, registers, an Arithmetic and Logic Unit (ALU), an instruction address register, and an instruction register. Some of its’ components are as follows:

* RAM: Location to store memory
* Registers: Temporarily store and manipulate values
* ALU: Perform mathematical or logical operations
* Instruction address register: Keep track of where we are in a program
* Instruction register: Store the current instruction

The CPU performs instruction in three phases:

* In the fetch phase, the CPU uses the instruction address register to access instruction from RAM, and then load that instruction into the instruction register. The instruction's location (address) in program memory is determined by the instruction pointer. After an instruction is fetched, the PC is incremented by the length of the instruction so that it will contain the address of the next instruction in the sequence
* In the decode phase, it decodes the instruction that was fetched to get the opcode and the RAM addresses to access. Often, one group of bits (that is, a "field") within the instruction, called the opcode, indicates which operation is to be performed, while the remaining fields usually provide supplemental information required for the operation, such as the operands. Those operands may be specified as a constant value (called an immediate value), or as the location of a value that may be a processor register or a memory address, as determined by some addressing mode
* The execute phase performs the instruction by configuring the ALU and other components based on the decoded opcode. Depending on the CPU architecture, this may consist of a single action or a sequence of actions. During each action, control signals electrically enable or disable various parts of the CPU so they can perform all or part of the desired operation.

The component that runs this process is called the control unit, which directs the flow of data and operations within the CPU, acting as the conductor of an orchestra. It tells the computer's memory, arithmetic and logic unit and input and output devices how to respond to the instructions that have been sent to the processor.

In order to run the computer clock forces the CPU to continue going through the phases by sending an electrical signal at regular intervals to advance internal operations. The clock speed, measured in Hertz, determines the frequency at which the CPU executes cycles. CPUs can be overclocked (increased clock speed) or underclocked (decreased clock speed) to optimize performance or save power.

**Deep Learning**

Classical machine learning involves defining functions to predict a label value (y) based on input features (x), typically using algorithms like logistic regression to determine coefficients. Deep learning is a class of machine learning algorithms that uses multiple layers to progressively extract higher-level features from the raw input.

Deep learning mimics the human brain by using artificial neurons with weighted inputs (x), bias (b), and an activation function to determine firing or non-firing (y). This can be represented by the equation, y = f(x, w, b), where:

* Activation function: Some function f(x, w, b) that outputs some value y to determine fire/non-firing. Often use a function that maps y to a value between 0 and 1.
* x: Inputs of the neurons
* w: Weights that is applied to input
* b: Bias
* y: Whether or not the neuron fires.

These artificial neurons are interconnected in layers, with weighted connections between neurons allowing for complex function approximation. A deep learning neural network has many interconnected layers, known as hidden layers, until the output is connected to an output layer, where the number of neurons is the same as the number of classes.

In deep learning, each level learns to transform its input data into a slightly more abstract and composite representation. In an image recognition application, the raw input may be a matrix of pixels; the first representational layer may abstract the pixels and encode edges; the second layer may compose and encode arrangements of edges; the third layer may encode a nose and eyes; and the fourth layer may recognize that the image contains a face. Importantly, a deep learning process can learn which features to optimally place in which level on its own. This does not eliminate the need for hand-tuning; for example, varying numbers of layers and layer sizes can provide different degrees of abstraction.

The loss function measures the difference between predicted probabilities and known labels, indicating the amount of error in the model. We want to use this loss function to collect the gradient in order to calculate how much you need to adjust the model's weight and biases to reduce the amount of loss. The amount you adjust each variable by is called the learning rate. A low learning rate results in small adjustments, meaning we might have to do a lot of iterations to minimize the error. Too large of a learning rate results in large adjustments, so we might step too far and not find the minimum part of the gradient

The gradient of the loss function guides adjustments to weights and biases to reduce loss through a process known as backpropagation. Backpropagation is a gradient estimation method that utilizes the chain rule to determine how best to adjust the weights and biases to minimize the loss function which involves. Training a neural network involves feeding input data with known labels and iteratively performing backpropagation to adjust weights and biases to minimize loss through epochs (cycles of training and validation). Since GPUs are optimized for the linear algebra calculations used in training deep learning models, it makes them commonly used for this purpose.

**Linux**

Today, Linux is prevalent, with over 850,000 Android phones and 700,000 TVs activated daily. Because of the dominance of Linux-based Android on smartphones, Linux, including Android, has the largest installed base of all general-purpose operating systems as of May 2022. Linux is the leading operating system on servers (over 96.4% of the top one million web servers' operating systems are Linux), leads other big iron systems such as mainframe computers, and is used on all of the world's 500 fastest supercomputers (as of November 2017, having gradually displaced all competitors). Linux powers 80% of financial trades and 90% of the world's supercomputers. Applications such as Google, Twitter, Facebook, and Amazon rely on Linux for their operations.

Linux development is collaborative, involving thousands of developers from hundreds of companies. Since 2005, 8,000 developers have contributed 15 million lines of code to the Linux kernel. The development of Linux is rapid, with major kernel releases every 2-3 months. This is made possible by a unique collaborative development process. When submitting code to the Linux kernel, developers break changes into individual units called patches. A patch describes the lines that need to be changed, added or removed from the source code.

Developers post their patches to the relevant mailing lists, where other developers can reply with feedback. When the patch is close to being release-ready, it is accepted by a senior Linux kernel developer or maintainer, who manages one or more of the 100 different sections of the kernel. This unique collaborative development process ensures code quality and security.

Linus Torvalds is the lead maintainer for the Linux kernel and guides its development, while Greg Kroah-Hartman is the lead maintainer for the stable branch. Zoë Kooyman is the executive director of the Free Software Foundation, which in turn supports the GNU components. Finally, individuals and corporations develop third-party non-GNU components. These third-party components comprise a vast body of work and may include both kernel modules and user applications and libraries.

**Early Programming**

Computers today use complex circuits in order to perform all sorts of tasks, such as storing and retrieving values, performing operations and processing a series of instructions within a program. In order for the computer to be able to do all the operations, they require the programs that call the operations be loaded into computer memory to be executed.

The need to program machines existed way before the development of computers. One of the first implementations of programming actually begins with the Jacquard’s programmable textile loom that utilizes punch cards to determine if a specific thread was held high or low in the loom, such that the cross thread passed above or below the thread. To apply variations to the design, these punch cards were arranged in long chains, forming a sequence of commands for the loom, which matches the ideas of the computer programming we know today. Punched cards turned out to be a cheap, reliable and fairly human-readable way to store data. Punched cards would also be used later on to help tabulate the 1890 US Census. The only problem with this system is that it only serves one purpose which is to tabulate.

Over time, these business machines grew in capability, adding arithmetic features as well as becoming capable of making simple decisions about when to perform certain operations. These operations were asked by the programmer through the control paneer, which in that time were called the plug boards. Using plug boards, programmers can plug cable to different sockets in order to pass values and signals between different parts of the machines, however, this requires the programmer to rewire the system every time a different operation wants to be done. To relieve this issue, swappable plug boards were created in order to ease the changing of operations without rewiring.

Stored-program computers emerged in the late 1940s, storing programs and data in memory for easy access and modification. Electronic memory enabled the von Neumann architecture, combining data and instructions in a single shared memory. The von Neumann architecture standardized a unified structure of these computers, including a central processing unit containing an ALU, data registers, instruction register, instruction address registers and memory to store data and instructions.

Even with electronic memory, punch cards remained a common way to load programs until the 1980s. Around 1960, the first mainframes—general purpose computers—were developed, although they could only be operated by professionals and the cost was extreme. The data and instructions were input by punch cards, meaning that no input could be added while the program was running. Other methods like punched paper tape and control panels with switches and buttons were also used. Another common way to control computers was through panel programming. In panel programming, switches and buttons were used to input data and programs, common on home computers of the 1950s and 60s.

Programming these early computers was difficult and time-consuming, requiring expertise in hardware and machine code. The need for a simpler way to program computers led to the development of programming languages. Programming languages differ from natural languages in that natural languages are only used for interaction between people, while programming languages also allow humans to communicate instructions to machines. After the invention of the microprocessor, computers in the 1970s became dramatically cheaper. New computers also allowed more user interaction, which was supported by newer programming languages.

**History of Computers**

Computers have become essential for various aspects of modern life, from infrastructure operation to scientific advancement. Computing devices have evolved over time, from the abacus and astrolabe to electronic computers. Initially a job title for human calculators, the term "computer" gradually shifted to refer to devices in the 1800s.

One of the more notable of these devices was Gottfried Leibniz's Step Reckoner which worked as an automated mechanical calculator. He attempted to create a machine that could be used not only for addition and subtraction but would use a moveable carriage to enable multiplication and division, however, Leibniz did not incorporate a fully successful carry mechanism. This design was one of the first that could do all 4 operations and would become used for the next three centuries for calculator design.

Unfortunately, even with mechanical calculators, most real-world problems required many steps of computations before determining an answer which could be time-consuming to generate. Before the 20th century, people performed computations through pre-computed tables.These tables could be used for tasks like finding square roots while being faster and more accurate than hand calculations. This was especially important for military applications like artillery firing where calculations needed to be both fast and accurate.

The problem with this is that when the design of the machine is changed, a whole-new pre-computed table had to be created which would also be very time-consuming and led to errors. Charles Babbage, an English mechanical engineer and polymath, originated the concept of a programmable computer. Often regarded as the "father of the computer", he conceptualized and invented the first mechanical computer in the early 19th century. After working on his revolutionary difference engine, he realized that a much more general design, an Analytical Engine, was possible.

The Analytical Engine was a general-purpose computer, not limited to one task, could be given data and run operations in sequence, and had a memory and a primitive printer. Like the Difference Engine, his idea was ahead of its time and was never fully built but inspired future generations. Ada Lovelace, who most consider the world's first programmer, wrote hypothetical programs for the Analytical Engine. The Analytical Engine would go on to inspire the first generation of computer scientists who incorporated these ideas into their machines.

By the end of the 19th century, the computing devices only found use for special-purpose tasks in science and engineering, but rarely seen in business, government or domestic life. The first application of computing devices in such scenarios was through Herman Hollerith's Punch Card Tabulating machine, which was an electro-mechanical machine that used punch cards to efficiently process data for the 1890 US census. That census was processed two years faster than the prior census had been. Hollerith's company eventually became the core of IBM and electro-mechanical business machines became a huge success, transforming commerce and government, and eventually setting the stage for digital computers.

**History of Linux**

In 1991, Linus Torvalds created Linux, an open-source operating system based on the UNIX operating system. Torvalds began the development of the Linux kernel on MINIX and applications written for MINIX were also used on Linux. Later, Linux matured and further Linux kernel development took place on Linux systems. GNU applications also replaced all MINIX components, because it was advantageous to use the freely available code from the GNU Project with the fledgling operating system; code licensed under the GNU GPL can be reused in other computer programs as long as they also are released under the same or a compatible license. Torvalds initiated a switch from his original license, which prohibited commercial redistribution, to the GNU GPL.Torvald decided to release Linux under the GPL license granting users significant freedoms, including the ability to modify and redistribute the software.

The adoption of Linux in production environments, rather than being used only by hobbyists, started to take off first in the mid-1990s in the supercomputing community, where organizations such as NASA started to replace their increasingly expensive machines with clusters of inexpensive commodity computers running Linux. Over time, due to Torvald’s decision to release Linux under the GPL license, Linux gained global popularity and fueled the rise of a commercial ecosystem around it.

Companies like Red Hat and IBM invested heavily in Linux, leading to its widespread adoption. Commercial use began when Dell and IBM, followed by Hewlett-Packard, started offering Linux support to escape Microsoft's monopoly in the desktop operating system market.Today, Linux powers numerous critical systems, including stock exchanges, web servers, and supercomputers.

The Linux development community continues to thrive with thousands of contributors and regular releases. Linus Torvalds, the creator of Linux, remains actively involved in its development from his home office in Portland, Oregon. Linus Torvalds is the lead maintainer for the Linux kernel and guides its development and individuals and corporations develop third-party non-GNU components. These third-party components comprise a vast body of work and may include both kernel modules and user applications and libraries.

**WebAudio API**

The Web Audio API specification developed by W3C describes a high-level JavaScript API for processing and synthesizing audio in web applications. The primary paradigm is of an audio routing graph, where a number of AudioNode objects are connected together to define the overall audio rendering.

The Web Audio API allows developers to perform tasks such as:

* Process and synthesize audio in web applications
* Load, play, loop, and process sound samples
* Apply sound effects such as reverberation, delay, graphic equalizer, compressor, distortion, etc.
* Create real-time audio visualizations like dancing frequency graphs, animated waveforms that dance with the music, or generate music programmatically.

These capabilities make WebAudioAPI suitable for games and music applications. WebAudio API is available on web browsers such as Google Chrome, Firefox, Opera, Safari, Microsoft Edge and Opera GX and also available on Google Chrome, Firefox and Safari on mobile devices.

Another particular suitable multimedia feature is Track API. Using Track API, you will be able to synchronize video with other elements in the document, such as display a Google Map, an HTML description or a Wikipedia page beside the video, while it’s playing.

**Python**

Expressions are a combination of objects and operators that compute a value. Many expressions involve a boolean data type, which takes values True or False and must be capitalized. Not capitalizing will make it unrecognizable to Python and will throw an error.

Boolean operations ("or", "and", "not") take one or more boolean objects and return boolean results.

* "X Or Y" returns True if either the X or Y argument is True.
* "X And Y" returns True only if both X and Y arguments are True.
* "Not X" negates the value of its argument X, switching the truth value of the boolean.

Python provides eight comparison operations for various types, including numbers and sequences. Although they are commonly used for numeric types, we can apply them to other types as well. The result of a comparison will always return a boolean output.

* <: Less than
* <=: Less than or equal to
* >: greater than
* >=: greater than or equal to
* ==: Equal to
* !=: Not equal to
* is: Same object
* is not: Different object

The “is” and “is not” can test if two objects are one and the same, which is different from saying whether or not contents are the same. In Python, “==” compares by value. The “is” operator may be used to compare object identities (comparison by reference), and comparisons may be chained—for example, a <= b <= c.

In Python, a distinction between expressions and statements is rigidly enforced. Statements cannot be a part of an expression—so list and other comprehensions or lambda expressions, all being expressions, cannot contain statements. A particular case is that an assignment statement such as a = 1 cannot form part of the conditional expression of a conditional statement.

**Instruction Programs**

CPUs combine an ALU, control unit, memory, and clock to perform tasks based on instructions. What makes CPU’s powerful is that they are programmable, such that if you write a different set of instructions, it will perform a completely different task. This way, the CPU becomes a hardware that can be used by easy to modify software.

Instructions are stored in memory, allowing for easy program modification and control by software. From the point of view of the CPU, machine code is stored in RAM, but is typically also kept in a set of caches for performance reasons. There may be different caches for instructions and data, depending on the architecture.

Instructions are sequences of opcodes and addresses/registers that control CPU operations. Each opcode responds to a particular instruction within the computer’s defined instruction table and performs the given task utilizing the address or registers in the second half of the instruction sequence. Most instructions have one or more opcode fields that specify the basic instruction type (such as arithmetic, logical, jump, etc.), the operation (such as add or compare), and other fields that may give the type of the operand(s), the addressing mode(s), the addressing offset(s) or index, or the operand value itself (such constant operands contained in an instruction are called immediate). Every processor or processor family has its own instruction set. Instructions are patterns of bits, digits, or characters that correspond to machine commands. Thus, the instruction set is specific to a class of processors using (mostly) the same architecture.

Some examples of some basic instructions within the instruction table include LOAD, STORE, ADD, and SUBTRACT, enabling simple arithmetic operations. We also have other operations such as JUMP and JUMP\_NEGATIVE instructions that allow for program flow control and conditional execution. JUMP allows for the processor to update the instruction address register to a new address, possibly skipping some instructions while JUMP\_NEGATIVE does the same thing in the event that an ALU instruction outputs a negative result. There is also the HALT instruction which signifies the end of a program and prevents infinite loops.

It should be noted that we can store instructions and data in the same memory, as there is no difference fundamentally as they are all binary numbers. It is for this reason that the HALT instruction is important as to differentiate the instructions from the data. You should also keep in mind that the placement of JUMP or JUMP\_NEGATIVE operations should be carefully considered, as it is possible to create a sequence of instructions such that we have an infinite loop. To break such a loop, we need another condition jump to exit the looped sequence in order to proceed with the program.

In these examples, we are working with a simple CPU that uses 8-bit instructions. This leads to complications as we are limited to 16 lines of instruction and data. Modern CPUs use larger instruction lengths (32-64 bits) and variable-length instructions, increasing instruction capacity and flexibility. How the patterns are organized varies with the particular architecture and type of instruction.

**Linear Regression**

Linear regression plays an important role in the subfield of artificial intelligence known as machine learning. The linear regression algorithm is one of the fundamental supervised machine-learning algorithms due to its relative simplicity and well-known properties. Linear regression can be used to fit a predictive model to an observed data set of values of the response and explanatory variables.

This video introduces linear regression as the first learning algorithm, showcasing its process in supervised learning. In supervised learning, we are training a model using a dataset such that each example of the data is given the “right answer” to find the function that we want to discover.

For example, suppose your friend is looking to sell their house and looking to see how much their house is worth. What they have is a dataset consisting of labeled pairs of the sizes of houses and its associated price. A supervised learning algorithm will learn from these pairs and will try to learn a mapping from the sizes of a house to the price of a house from the dataset. The model can then be used to predict the value of your friend’s house based on the size of their house.

This problem is also known as a regression problem, as the term regression refers to the fact that we are trying to predict real-valued output.

In supervised learning, we have a training set (in this example, it is the set of the prices of the house) and our job is to learn from this data how to predict the outputs (the house prices). For this, we are going to introduce some notation:

* “m” will refer to the number of training examples
* “x” will refer to an input variables/features
* “y” will refer to the output/target variables
* (x, y) will refer to a single training example (input/output pair)
* (x(i), y(i)) will refer to the i-th specific training example

In supervised learning, we start off with a training set and we feed that to our learning algorithm. It is the job of the learning algorithm to then output a function which by convection is usually denoted “h” where “h” stands for hypothesis. The hypothesis is a function, in the housing example, is to take the size of the house(x) and try to output the estimated value (y). In summary, “h” is a function that maps x’s to y’s.

In linear regression, the hypothesis function is assumed to be a linear relationship between X and Y. We can represent this relationship as function h(x) = Θ0 + Θ1 \* x, where Θ0 and Θ1 are parameters learned by the learning algorithm. Linear regression with one variable is also known as simple linear regression or univariate linear regression.