Learning

Learning enables people to improve their performance in an area of study—whether it is dentistry or violin playing. Students in dental school become more proficient at repairing teeth whereas a violinist attending the Juilliard School in New York City is likely to play a Mozart violin concerto with greater artistry after several years of training. Similarly, machine learning is the process whereby a computer distills meaning by exposure to training data. An agent is learning if it improves its performance on future tasks after making observations about the world.

Can machines think?

If we were to discover algorithms wherein computers were enabled to perform the analytical reasoning entailed in learning (beyond the application of deductive principles), this would go a long way toward resolving this question—as most people believe that learning is an essential component of thinking.

Why would we want an agent to learn? If the design of the agent can be improved, why wouldn't the designers just program in that improvement to begin with? There are three main reasons.

- The designers cannot anticipate all possible situations that the agent might find itself in. For example, a robot designed to navigate mazes must learn the layout of each new maze it encounters.
- The designers cannot anticipate all changes over time; a program designed to predict tomorrow's stock market prices must learn to adapt when conditions change from boom to bust.
- Sometimes human programmers have no idea how to program a solution themselves. For example, most people are good at recognizing the faces of family members, but even the best programmers are unable to program a computer to accomplish that task, except by using learning algorithms.

Feedback

Feedback is information given to the learner about the learner's performance relative to learning goals or outcomes. It should aim to (and be capable of producing) improvement in learning.

Feedback redirects or refocuses the learner's actions to achieve a goal, by aligning effort and activity with an outcome. It can be about the output or outcome of the task, the process of the task, the systems management of their learning or self-regulation.

Types of Learning

On the basis of feedback, there are three types of learning

Supervised learning

In supervised learning the agent observes some example input–output pairs and learns a function that maps from input to output. In the taxi driving agent, the inputs are percepts and the output are provided by a teacher who says "Brake!" or "Turn left." Once enough examples are provided to the agent, the theory of braking which is a function from states and braking actions to stopping distance in feet is developed. Then the agent use this function to automatically know when to brake or turn left and so on.

Unsupervised learning

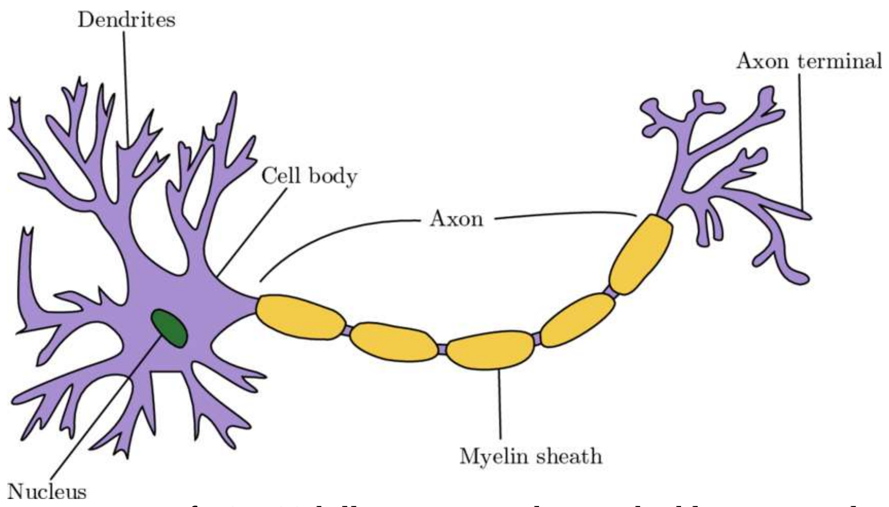
In unsupervised learning the agent learns patterns in the input even though no explicit feedback is supplied. The most common unsupervised learning task is clustering: detecting potentially useful clusters of input examples. For example, a taxi agent might gradually develop a concept of "good traffic days" and "bad traffic days" without ever being given labeled examples of each by a teacher.

Reinforcement learning

In reinforcement learning the agent learns from a series of reinforcements—rewards or punishments. For example, the lack of a tip at the end of the journey gives the taxi agent an indication that it did something wrong. The two points for a win at the end of a chess game tells the agent it did something right. It is up to the agent to decide which of the actions prior to the reinforcement were most responsible for it.

Neural Network

Whenever we wish to design a system to perform some activity, it is a good idea to begin by asking if a solution already exists in nature. For example, you wanted to design an artificial flying machine (an airplane). You would observe that natural lying "machines" do in fact exist (birds). Your airplane design would probably incorporate two large wings. So it seems natural that if you want to design artificially intelligent systems (as we do) that you begin your studies by analyzing one of the most naturally intelligent systems on this planet—the human brain and nervous system.



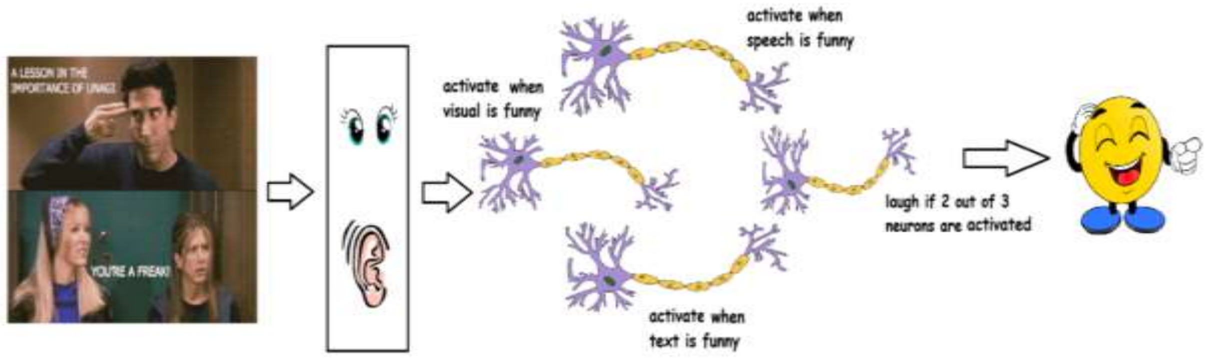
The human brain consists of 10–100 billion neurons that are highly connected to one another. Some neurons communicate with several or perhaps several dozen neighboring neurons, whereas others have thousands of neurons with which they share information. Drawing inspiration from this natural paradigm, researchers have designed artificial neural networks (ANN) over the past decades. Applications have ranged from stock market forecasting to autonomous control of automobiles.

The human brain is an adaptive system that must respond to the vagaries of existence. Learning takes place by modifying the strengths of connections between neurons. In a similar manner, artificial neural network weights must change to take on this same adaptability. In one ANN paradigm— supervised learning—learning rules assume responsibility for this task by comparing a network's performance against desired responses and then modifying the system's weights accordingly.

RUDIMENTS OF ARTIFICIAL NEURAL NETWORKS

McCulloch and Pitts developed the first model for artificial neurons. They were attempting to understand (and to simulate) the behavior of animal nervous systems. Present day biologists and neurologists do understand how individual neurons communicate with one another inside a living being. Animal nervous systems are composed of thousands or millions of these interconnected cells; in humans, it is billions. The way in which parallel collections of neurons form functional units, however, is still a mystery.

Electrical signals flow into the cell body via dendrites, which are hair-like filaments. The cell body (or soma) is where "processing" occurs. When sufficient excitation is present, the neuron fires; in other words, it sends a small electrical signal (measured in milliwatts) down the cable-like protrusion known as an axon. A neuron will usually have a single axon but will possess many dendrites. By sufficient excitation we mean in excess of some predetermined threshold. The electrical signal flows through the axon until it reaches the end bulb. The axon-dendrite (or axon-soma or axon-axon) contact between an end bulb and a cell it encroaches on is called a synapse.


A LESSON IN THE IMPORTANCE OF UNAGI
YOU'RE A FREAK!
activate when visual is funny
activate when speech is funny
activate when text is funny
laugh if 2 out of 3 neurons are activated

There is actually a small gap between two neurons (that almost touch)—this is called a synaptic gap. This gap is laden with a conductive fluid that permits the flow of interneuronal electrical signals. Brain hormones (or ingested drugs such as caffeine) affect the degree of conductivity that is present. There are four elements that AI has adopted from this biological model:

Biological model	**Artificial neurons**
• Cell body	• Cell body
• Axon	• Output channel
• Dendrites	• Input channel
• Synapses	• Weights

As seen above, real-valued weights play the role of synapses. The value of a weight reflects the conductive level of a biological synapse and serves to mediate the degree of influence that one neuron has on another. An abstract neuron (sometimes called a unit or node or just a neuron) is depicted in Figure alongside.

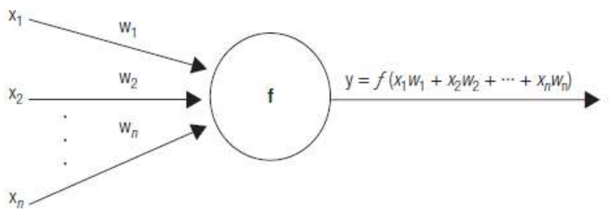


Figure
A model of an abstract neuron.

The input to a neuron is a real-valued vector $\bar{x} = (x_{1,} x_2, \cdots, x_n)$ with n components. A weight vector$\bar{w} = (w_{1,} w_2, \cdots, w_n)$ also real-valued, is the counterpart to a synapse in a biological neuron. These weights govern the effect that the inputs will have on the unit. The body of a neuron computes a primitive function f. Finally, the output of this unit y is equal to the function f applied to the dot product of $\bar{x}$ with $\bar{w}$. More generally, the network computes some function g of the inputs and weights. Figure alongside illustrates the more general situation. Here, g is a function of the inputs $\bar{x}$ and $\bar{w}$ and f is the output or activation function.

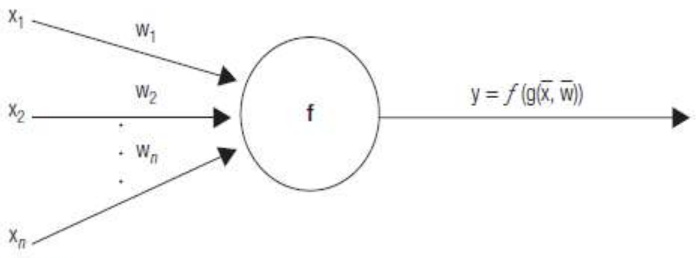


Figure
A generic neuron.

An ANN can be viewed as a black box as seen in Figure alongside. Certain input vectors $\bar{x}$ should produce specific outputs $\bar{y}$. To accomplish this feat, the network must adjust its weights in a self-organizing process.

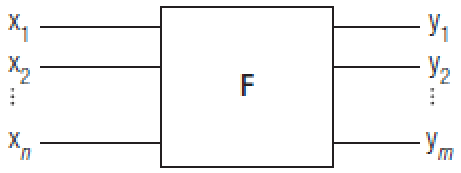


Figure
A neural network as a black box.

McCULLOCH-PITTS NETWORK

Working from the beginnings of neuroscience, Warren McCulloch and Walter Pitts in their 1943 paper, "A Logical Calculus of Ideas Immanent in Nervous Activity," contended that neurons with a binary threshold activation function were analogous to first order logic sentences. The basic McCulloch and Pitts neuron is depicted in figure alongside.

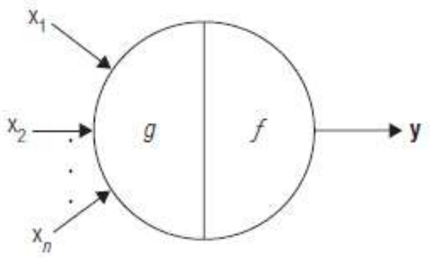


Figure
Diagram of a McCulloch-Pitts neuron.

The McCulloch-Pitts neuron worked by inputting either a 1 or 0 for each of the inputs, where 1 represented true and 0 false. Edges are either excitatory or inhibitory. The inhibitory edges are marked with a small circle just near the unit. The threshold value is is denoted by θ.

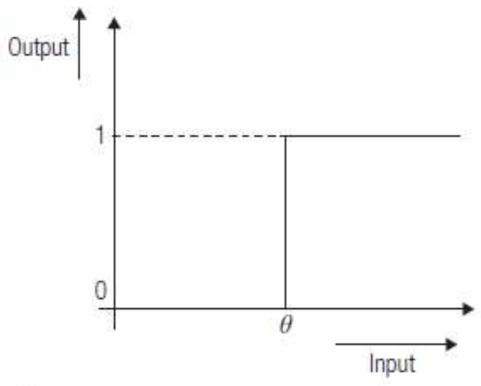


Figure
The step function for a McCulloch-Pitts neuron with threshold θ

Threshold was given a real value, which would allow for a 0 or 1 output if the threshold was met or exceeded.

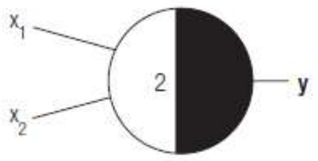
The inputs $x_{1,}x_2, \cdots, x_n$ enter the neuron through n excitatory edges. There can also be inputs $v_{1,}v_2, \cdots, v_m$ entering the unit through m inhibitory edges; if any inhibitory inputs are present, then the neuron is inhibited and its output y will equal 0.

Otherwise, the total excitation $g(\bar{x}) = x_1 + x_2 + \cdots + x_n$ If $g(\bar{x}) \geq \theta$ then the unit fires, and y = 1. The activation function f which produces the output of this unit is a step (or threshold) function. Note that when the total excitation $g(x) < \theta$, the output is 0. This type of activation function is called a step function.

Boolean Functions Using McCulloch-Pitts Neuron

McCulloch-Pitts model can be used to represent Boolean functions. Here the inputs are all Boolean and the output is also Boolean. So essentially, the neuron is just trying to learn a Boolean function. A lot of Boolean decision problems can be cast into this, based on appropriate input variables.

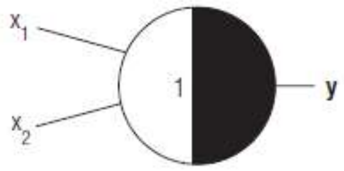
AND gate



X_1	X_2	ΣX_i	y
0	0	0	0
0	1	1	0
1	0	1	0
1	1	2	1

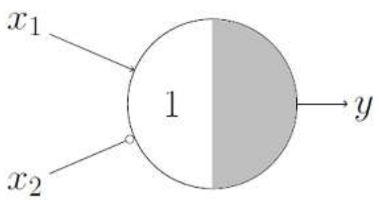
Here, the threshold value is set to 2 (it is always equal to number of inputs for AND gate). So, whenever ΣX_i is less than 2, the output is 0. In case of ΣX_i being equal to or greater than 2, the ouptut of neurone is 1. The task of producing output from input is performed by a function called activation function. The activation function used in McCulloch Pitts neurone is called step function. The neurone works in similar way for all other gates implemention as well.

OR gate



X_1	X_2	ΣX_i	y
0	0	0	0
0	1	1	1
1	0	1	1
1	1	2	1

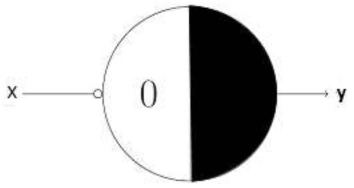
A Function With An Inhibitory Input



X_1	X_2	ΣX_i	y
0	0	0	0
0	1	-	0
1	0	1	1
1	1	-	0

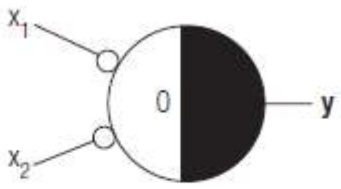
This might look like a tricky one but it’s really not. Here, we have an inhibitory input i.e., x_2 so whenever x_2 is 1, the output will be 0 regardless of the value of x_1.

NOT gate



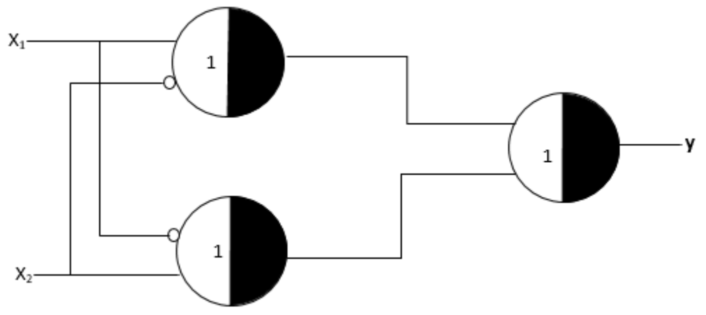
X	ΣX_i	y
0	0	1
1	-	0

NOR gate



X_1	X_2	ΣX_i	y
0	0	0	1
0	1	-	0
1	0	-	0
1	1	-	0

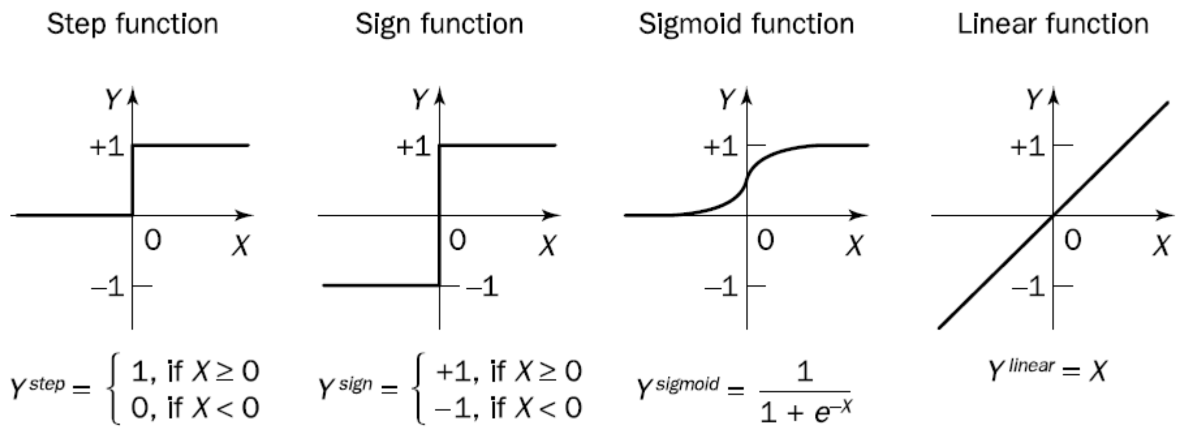
XOR gate



X_1	X_2	ΣX_i	y
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Is the step function the only activation function used by neurons?

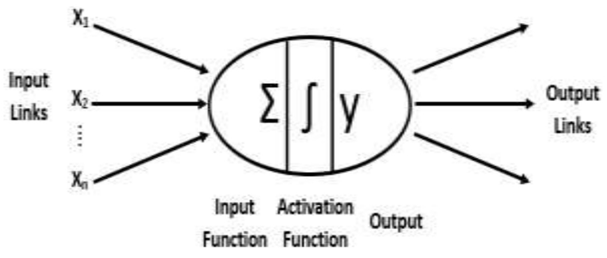
Many activation functions have been tested, but only a few have found practical applications. Four common choices – the step, sign, linear and sigmoid function are illustrated in Figure below. The step and sign activation functions, also called hard limit functions, are often used in decision-making neurons for classification and pattern recognition tasks. The sigmoid function transforms the input, which can have any value between plus and minus infinity, into a reasonable value in the range between 0 and 1. Neurons with this function are used in the back-propagation networks. The linear activation function provides an output equal to the neuron weighted input. Neurons with the linear function are often used for linear approximation



$$Y^{step} = \begin{cases} 1, \text{ if } X \geq 0 \\ 0, \text{ if } X < 0 \end{cases} \quad Y^{sign} = \begin{cases} +1, \text{ if } X \geq 0 \\ -1, \text{ if } X < 0 \end{cases} \quad Y^{sigmoid} = \frac{1}{1+e^{-X}} \quad Y^{linear} = X$$

Figure Activation functions of a neuron

Neural Network Architecture



Neural networks are composed of nodes or units connected by directed links. A link from unit i to unit j serves to propagate the activation a_i from i to j. Each link also has a numeric weight $w_{i,j}$ associated with it, which determines the strength and sign of the connection. Each unit j first computes a weighted sum of its inputs:

$$in_j = \sum w_{ij} a_i$$

Then it applies an activation function g to this sum to derive the output:

$$a_j = g(in_j) = g\left(\sum w_{ij} a_i\right)$$

The activation function g is typically either a hard threshold, in which case the unit is called a perceptron, or a logistic function, in which case the term sigmoid perceptron is sometimes used.

Having decided on the mathematical model for individual “neurons,” the next task is to connect them together to form a network. There are two fundamentally distinct ways to do this: Feed forward network and recurrent network. A recurrent network, on the other hand, feeds its outputs back into its own inputs.

A neural network can consist of three types of nodes:

Input Nodes - The Input nodes provide information from the outside world to the network and are together referred to as the "Input Layer". No computation is performed in any of the Input nodes - they just pass on the information to the hidden nodes.

Hidden Nodes - The Hidden nodes have no direct connection with the outside world (hence the name "hidden"). They perform computations and transfer information from the input nodes to the output nodes. A collection of hidden nodes forms a "Hidden Layer". While a feedforward network will only have a single input layer and a single output layer, it can have zero or multiple Hidden Layers.

Output Nodes - The Output nodes are collectively referred to as the "Output Layer" and are responsible for computations and transferring information from the network to the outside world.

Supervised (Error based) Learning

The task of supervised learning is this:

> Given a training set of N example input–output pairs
>
> *$(x_1, y_1), (x_2, y_2), \ldots (x_N, y_N)$* ,
>
> where each y_j was generated by an unknown function y = f(x), discover a function *h* that approximates the true function f.

Here x and y can be any value; they need not be numbers. The function h is a hypothesis. Learning is a search through the space of possible hypotheses for one that will perform well, even on new examples beyond the training set. To measure the accuracy of a hypothesis we give it a test set of examples that are distinct from the training set.

Feed Forward Neural Networks

A feed-forward network has connections only in one direction—that is, it forms a directed acyclic graph. Every node receives input from “upstream” nodes and delivers output to “downstream” nodes; there are no loops. A feed-forward network represents a function of its current input; thus, it has no internal state other than the weights themselves. Feed-forward networks are usually arranged in layers, such that each unit receives input only from units in the immediately preceding layer. Feed forward neural network may be single-layer network or multi-layer network.

Two types of feedforward networks are given below:

Single-layer feed-forward neural networks (perceptron)

A network with all the inputs connected directly to the outputs is called a single-layer neural network, or a perceptron network. In this type of network, we have only two layers input layer and output layer but input layer does not count because no computation performed in this layer. Output layer is formed when different weights are applied on input nodes and the cumulative effect per node is taken. After this the neurons collectively give the output layer compute the output signals.

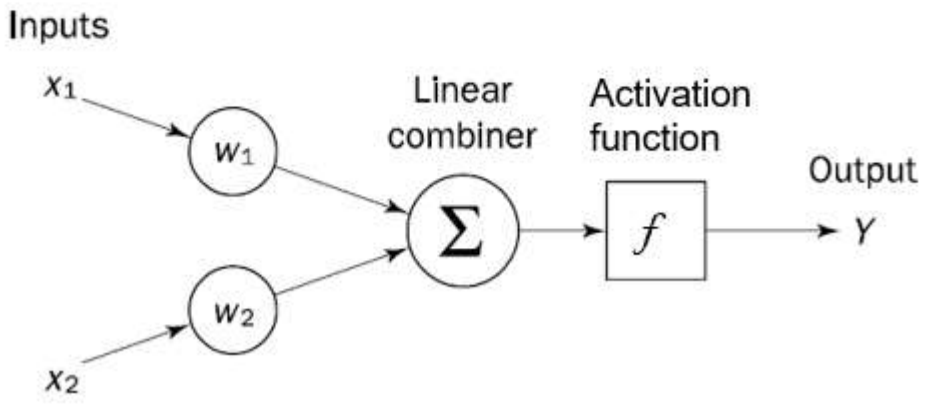
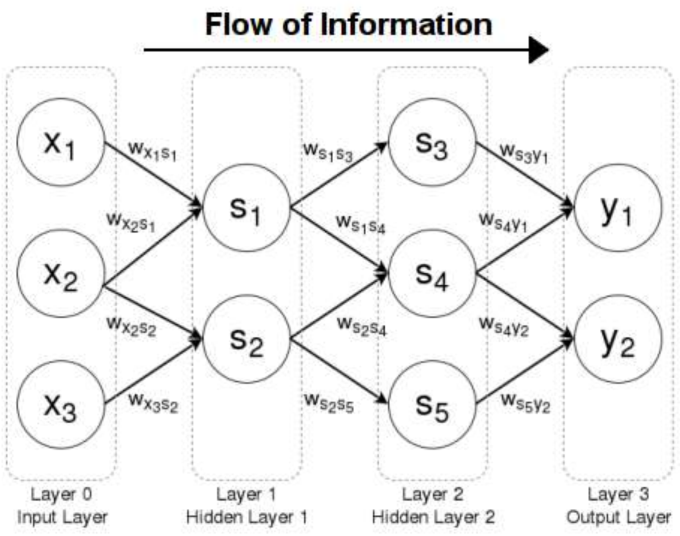


Figure Single-layer two-input perceptron

Multilayer feed forward network

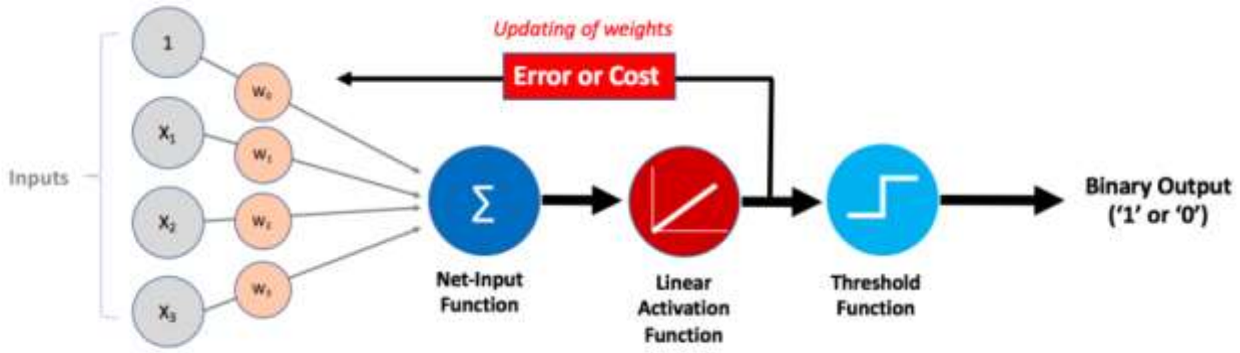
A multilayer perceptron is a feedforward neural network with one or more hidden layers. Typically, the network consists of an input layer of source neurons, at least one middle or hidden layer of computational neurons, and an output layer of computational neurons. The input signals are propagated in a forward direction on a layer-by-layer basis. A multilayer perceptron with two hidden layers is shown in Figure alongside.



Each layer in a multilayer neural network has its own specific function. The input layer accepts input signals from the outside world and redistributes these signals to all neurons in the hidden layer. Actually, the input layer rarely includes computing neurons, and thus does not process input patterns. The output layer accepts output signals, or in other words a stimulus pattern, from the hidden layer and establishes the output pattern of the entire network. Neurons in the hidden layer detect the features; the weights of the neurons represent the features hidden in the input patterns. These features are then used by the output layer in determining the output pattern.

ADALINE (ADAptive LInear NEurone)

ADALINE, as like Perceptron, also mimics a neuron in the human brain. Adaline is a single layer neural network with multiple nodes where each node accepts multiple inputs and generates one output. The difference between Adaline and the standard (McCulloch–Pitts) perceptron is that in the learning phase, the weights are adjusted according to the weighted sum of the inputs (the net).



The following represents the working of Adaline machine learning algorithm based on the above diagram:

- Input signals of different strength (weights) get combined / added in order to be fed into the activation function. The combined input or sum of weighted inputs can also be called as net input.
- Net input is fed into linear activation function.
- The output of activation function (same as net input owing to identity function) is used to calculate the change in weights related to different inputs which will be updated to learn new weights. There is a feedback loop that is used to calculate error or cost
- The prediction made by Adaline neuron is done in the same manner as in case of Perceptron. The output of activation function, which is net input is compared with 0 and the output is 1 or 0 depending upon whether the net input is greater than or equal to 0.

Given the following variables :

x is the input vector

w is the weight vector

n is the number of inputs

θ some constant

y is the output of the model

then, the output is calculated as:

$$y = \sum_{j=1}^{n} x_j w_j + \theta$$

Learning Algorithm

Let us assume:

η is the learning rate (some positive constant)

y is the output of the model

o is the target (desired) output

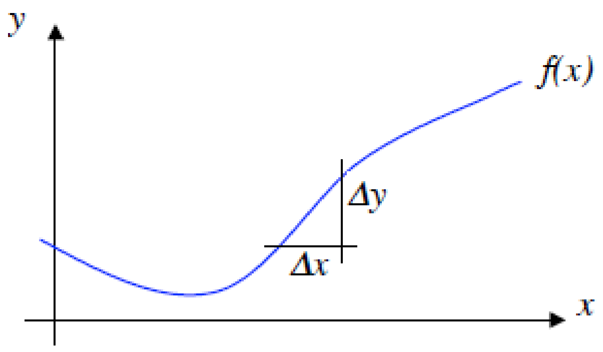
then the weights are updated as follows

$$w \leftarrow w + \eta(o - y)x$$

Here, $(o - y)$ is called error (E)

Gradient Descent Learning

Gradient descent is a calculus-based method used to find the minimum of a function. Suppose that the variable y depends on a single variable x, or y = f (x).



The gradient of f(x), at a particular value of x, is the rate of change of f(x) as we change x, and that can be approximated by Δy/Δx for small Δx. It can be written exactly as

$$\frac{\partial f(x)}{\partial x} = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

which is known as the partial derivative of f(x) with respect to x.

If we want to change the value of x to minimize a function f(x), what we need to do depends on the gradient of f(x) at the current value of x. There are three cases:

If $\partial f/\partial x > 0$ then f(x) increases as x increases so we should decrease x

If $\partial f/\partial x < 0$ then f(x) decreases as x increases so we should increase x

If $\partial f/\partial x = 0$ then f(x) is at a maximum or minimum so we should not change x

In summary, we can decrease f(x) by changing x by the amount:

$$\Delta x = x_{new} - x_{old} = -\eta \frac{\partial f}{\partial x}$$

where η is a small positive constant specifying how much we change x by, and the derivative $\partial f/\partial x$ tells us which direction to go in. If we repeatedly use this equation, f(x) will (assuming η is sufficiently small) keep descending towards a minimum, and hence this procedure is known as gradient descent minimization.

Stochastic Learning

The stochastic nature of machine learning algorithms is an important foundational concept in machine learning and is required to be understand in order to effectively interpret the behavior of many predictive models.

Stochastic refers to a variable process where the outcome involves some randomness and has some uncertainty. It is a mathematical term and is closely related to "randomness" and "probabilistic" and can be contrasted to the idea of "deterministic." Many machine learning algorithms and models are described in terms of being stochastic. This is because many optimization and learning algorithms both must operate in stochastic domains and because some algorithms make use of randomness or probabilistic decisions.

Stochastic gradient descent optimizes the parameters of a model, such as an artificial neural network, that involves randomly shuffling the training dataset before each iteration that causes different orders of updates to the model parameters. In addition, model weights in a neural network are often initialized to a random starting point. These algorithms make use of randomness during the process of constructing a model from the training data which has the effect of fitting a different model each time same algorithm is run on the same data.

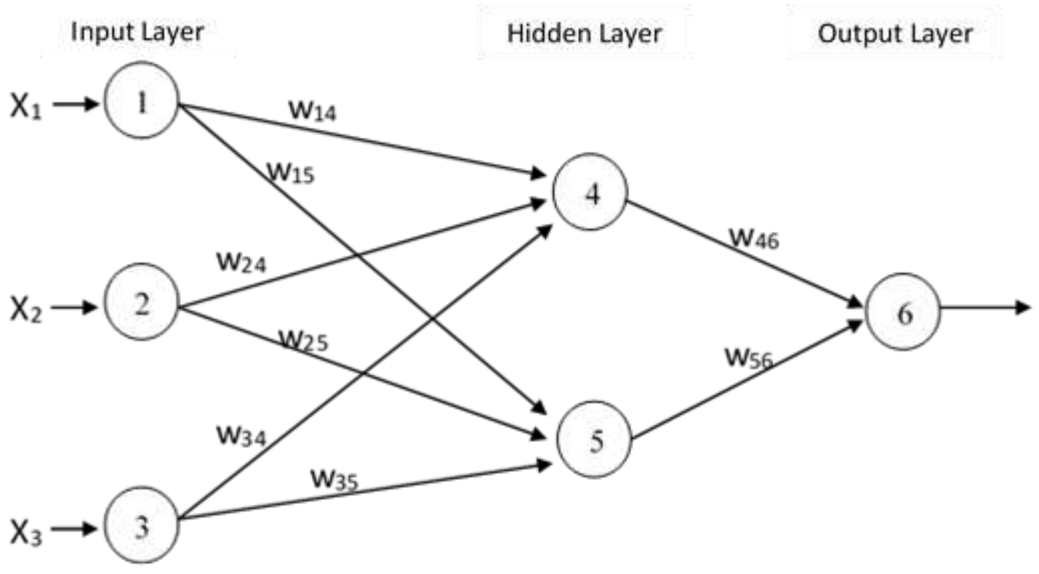
Backpropagation Algorithm

Backpropagation (backward propagation) is an important mathematical tool for improving the accuracy of predictions in data mining and machine learning. It is the method of fine-tuning the weights of a neural network based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows you to reduce error rates and make the model reliable by increasing its generalization. The algorithm gets its name because the weights are updated backwards, from output towards input.

The difficulty of understanding exactly how changing weights and biases affects the overall behavior of an artificial neural network was one factor that held back wider application of neural network applications, arguably until the early 2000s when computers provided the necessary insight. Today, backpropagation algorithms have practical applications in many areas of artificial intelligence (AI), including optical character recognition (OCR), natural language processing (NLP) and image processing. Because backpropagation requires a known, desired output for each input value in order to calculate the loss function gradient, it is usually classified as a type of supervised machine learning.

The back-propagation algorithm for learning in multilayer networks

inputs: examples, a set of examples, each with input vector x and output vector y network , a multilayer network with L layers, weights $w_{i,j}$, activation function g



```
1. for each weight w_{i,j} in network do
        $w_{i,j} \leftarrow small\ random\ number$
2. Propagate the inputs forward to compute the outputs
    2.1. for each node i in the input layer do
            $a_i \leftarrow x_i$
    2.2.  for all the nodes in do
            $in_j \leftarrow \sum_i w_{i,j}\, a_i$
            $a_j \leftarrow g(in_j)$
3. Propagate error backward from output layer to input layer
    3.1. for each node j in the output layer do
            $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
    3.2. For all other nodes
            $\Delta[i] \leftarrow g'(in_j) \sum_j w_{i,j}\, a_i\, \Delta[j]$
4. Update all the weights in network as
        $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
5. Repeat until stopping criterion is satisfied
```

Unsupervised Learning

Unsupervised learning refers to the use of artificial intelligence (AI) algorithms to identify patterns in data sets containing data points that are neither classified nor labeled. The algorithms are thus allowed to classify, label and/or group the data points contained within the data sets without having any external guidance in performing that task.

In other words, unsupervised learning allows the system to identify patterns within data sets on its own.In unsupervised learning, an AI system will group unsorted information according to similarities and differences even though there are no categories provided.

Unsupervised learning algorithms can perform more complex processing tasks than supervised learning systems. Additionally, subjecting a system to unsupervised learning is one way of testing AI.

Hebbian learning

The neuroscientific concept of Hebbian learning was introduced by Donald Hebb in his 1949 publication of The Organization of Behaviour. Also known as Hebb's Rule or Cell Assembly Theory, Hebbian Learning attempts to connect the psychological and neurological underpinnings of learning.

The basis of the theory is when our brains learn something new, neurons are activated and connected with other neurons, forming a neural network. These connections start off weak, but each time the stimulus is repeated, the connections grow stronger and stronger, and the action becomes more intuitive.

According to Donald Hebb, "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place on one or both cells such that A's efficiency as one of the cells firing B, is increased."

In more familiar terminology, that can be stated as the Hebbian Learning rule:

1. If two neurons on either side of a synapse (connection) are activated simultaneously (i.e. synchronously), then the strength of that synapse is selectively increased. Then, in the notation used for perceptrons, that can be written as the weight update:

$$\Delta w_{ij} = \eta . out_j . in_i$$

There is strong physiological evidence that this type of learning does take place in the region of the brain known as the hippocampus.

A good example is the act of learning to drive. When you start out, everything you do is incredibly deliberate. You remind yourself to turn on your indicator, to check your blind spot, and so on.

However, after years of experience, these processes become so automatic that you perform them without even thinking.

Competitive Learning

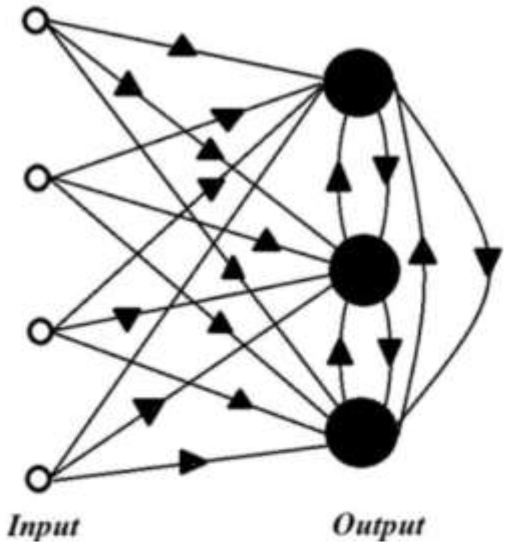
In competitive learning, as the name implies, the output neurons of a neural network compete among themselves to become active. Whereas in a neural network based on Hebbian learning several output neurons may be active simultaneously, in competitive learning only a single output neuron is active at any one time. It is this feature which makes competitive learning highly suited to discover statistically salient features which may be used to classify a set of input patterns.

There are three basic elements to a competitive learning rule:

i. A set of neurons which are all the same except for some randomly distributed synaptic weights, and which therefore, respond differently to a given get of input patterns.

ii. A limit imposed on the ‘strength’ of each neuron.

iii. A mechanism which permits the neurons to compete for the right to respond to a given subset of inputs, such that only one output neuron or only one neuron per group, is active (i.e., ‘on’) at a time. The neuron which wins the competition is called a winner-takes-all neuron.



Accordingly the individual neurons of the network learn to specialize on ensembles of similar patterns; in so doing they become feature detectors for different classes of input patterns.

In the simplest form of competitive learning, the neural network has a single layer of output neurons, each of which is fully connected to the input nodes. The network may include feedback connections among the neurons, as indicated in figure alongside. In the network architecture described herein, the feedback connections perform lateral inhibition, with each neuron tending to inhibit the neuron to which it is laterally connected. In contrast, the feed forward synaptic connections in the network all are excitatory.

For a neuron k to be the winning neuron, it's induced local field v_k for a specified input pattern x must be the largest among all the neurons in the network. The output signal y_k of winning neuron k is set equal to one; the output signals of all the neurons which lose the competition are set equal to zero. We thus write

$$y_k = \begin{cases} 1, if\ v_k for\ all\ j, j \neq k \\ 0, otherwise \end{cases}$$

where, the induced local field y_k represents the combined action of all the forward and feedback inputs to neuron k.

Let ω_{kj} denote the synaptic weight connecting input node j to neuron k. Suppose that each neuron is allotted a fixed amount of synaptic weight (i.e., all synaptic weights are positive), which is distributed among its input nodes that is, for all k

$$\sum_j \omega_{kj} = 1 \quad for\ all\ j$$

A neuron then learns by shifting synaptic weights from its inactive to active input nodes. If a neuron does not respond to a particular input pattern, no learning takes place in that neuron.

If a particular neuron wins the competition, each input node of that neuron relinquishes some proportion of its synaptic weight, and the weight relinquished is then distributed equally among the active input nodes. According to the standard competitive learning rule, the change $\Delta\omega_{kj}$ applied to synaptic weight ω_{kj} is defined by

$$\Delta\omega_{kj} = \begin{cases} \eta\left(x_j - w_{ij}\right), if\ neurone\ k\ wins\ the\ competition \\ 0, \quad if\ neurone\ k\ loses\ the\ competition \end{cases}$$

where, η is the learning rate parameter. This rule has the overall effect of moving the synaptic weight vector ω_k of winning neuron k towards the input pattern,v

Reinforcement Learning

Reinforcement learning in formal terms is a method of machine learning wherein the software agent learns to perform certain actions in an environment which lead it to maximum reward. It does so by exploration and exploitation of knowledge it learns by repeated trials of maximizing the reward.

Intuition to Reinforcement Learning

Let us try to understand the previously stated formal definition by means of an example -

Imagine you are supposed to cross an unknown field in the middle of a pitch black night without a torch. There can be pits and stones in the field, the position of those are unfamiliar to you. There's a simple rule - if you fall into a hole or hit a rock, you must start again from your initial point.

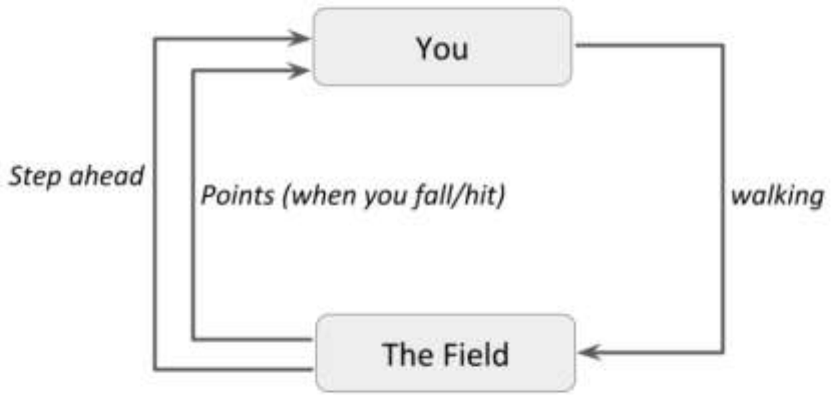
You start walking forward blindly, only counting the number of steps you take. After x steps, you fall into a pit. Your reward was x points since you walked that many steps.

You start again from your initial position, but after x steps, you take a detour either left/right and again move forward. You hit a stone after y steps. This time your reward was y which is greater than x. You decide to take this path again but with more caution.

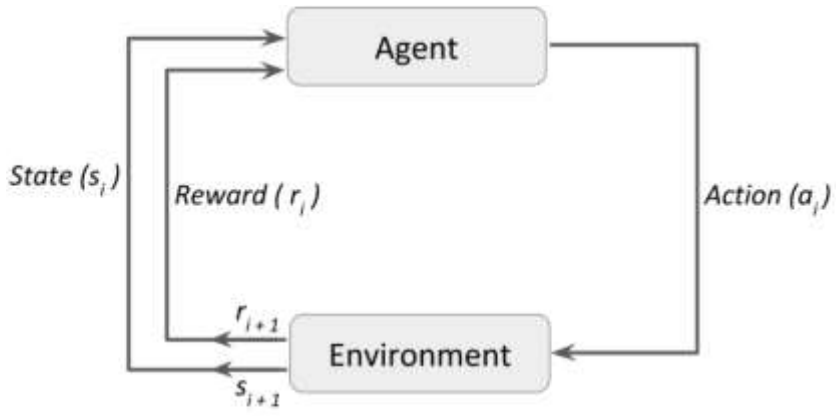
When you start again, you make a detour after x steps, another after y steps and manage to fall into another pit after z steps. This time the reward was z points which was greater than y, and you decide that this is a good path to take again.

You restart again, make the detours after x, y and z steps to reach the other side of the field. Thus, you've learned to cross the field without the need of light.

In the above example, you are the agent who is trying to walk across the field, which is the environment. Walking is the action the agent performs on the environment. The distance the agent walks acts as the reward. The agent tries to perform the action in such a way that the reward maximizes. This is how Reinforcement Learning works in a nutshell. The following figure puts it into a simple diagram –



And in the proper technical terms, and generalizing to fit more examples into it, the diagram becomes –



Genetic Algorithms

Genetic algorithm learning methods are based on models of natural adaptation and evolution. These learning systems improve their performance through processes which model population genetics and survival of the fittest. They have been studied since the early 1960s.

In GA, a solution is represented by a string. In canonical GA this string is binary, in other words, a sequence of 0s and 1s, though real numbers and other representations are possible. This string is often referred to as a chromosome.

Genetic algorithm systems stall with a fixed size population of data structures which are used to perform some given tasks. After requiring the structures to execute the specified tasks some number of times, the structures are rated on their performance, and a new generation of data structures is then created. The new generation is created by mating the higher performing structures to produce offspring. These offspring and their parents are then retained for the next generation while the poorer performing structures are discarded. The basic cycle is illustrated in Figure 17.8.

Mutations are also performed on the best performing structures to insure that the full space of possible structures is reachable. This process is repeated for a number of generations until the resultant population consists of only the highest performing structures.

Data structures which make up the population can represent rules or any other suitable types of knowledge structure. To illustrate the genetic aspects of the problem, assume for simplicity that the population of structures are fixed-length binary strings such as the eight bit string 11010001. An initial population of these eight-bit strings would be generated randomly or with the use of heuristics at time zero. These strings, which might be simple condition and action rules, would then be assigned some tasks to perform (like predicting the weather based on certain physical and geographic conditions or diagnosing a fault in a piece of equipment).

After multiple attempts at executing the tasks, each of the participating structures would be rated and tagged with a utility value u commensurate with its performance. The next population would then be generated using the higher performing structures as parents and the process would be repeated with the newly produced generation. After many generations the remaining population structures should perform the desired tasks well.

Mating between two strings is accomplished with the crossover operation which randomly selects a bit position in the eight-bit string and concatenates the head of one parent to the tail of the second parent to produce the offspring. Suppose the two parents are designated as xxxxxxxx and yyyyyyyy respectively, and suppose the third bit position has been selected as the crossover point (at the position of the colon in the structure xxx:xxxxx). After the crossover operation is applied, two offspring are then generated namely xxxyyyyy and yyyxxxxx. Such offspring and their parents are then used to make up the next generation of structures.

A second genetic operation often used is called inversion. Inversion is a transformation applied to a single string. A bit position is selected at random, and when applied to a structure, the inversion operation concatenates the tail of the string to the head of the same string. Thus, if the sixth position were selected ($x_1,x_2x_3x_4x_5x_6:x_7x_8$) the inverted string would be $x_7x_8x_1,x_2x_3x_4x_5x_6$.

A third operator, mutation, is used to insure that all locations of the rule space are reachable, that every potential rule in the rule space is available for evaluation. This insures that the selection process does not get caught in a local minimum. For example, it may happen that use of the crossover and inversion operators will only produce a set of structures that are better than all local neighbors but not optimal in a global sense. This can happen since crossover and inversion may not be able to produce some undiscovered structures. The mutation operator can overcome this by simply selecting any bit position in a string at random and changing it. This operator is typically used only infrequently to prevent random wandering in the search space.

Natural Language Processing

One of the oldest, most researched, and most demanding fields of AI is speech and language understanding. Any attempt at developing intelligent systems ultimately seems compelled to address what form of communication will be the standard. Language communication is often the preferred choice over, for example, graphical or data-based systems. The foundations for natural language understanding were established in the 1940s and 50s with work on finite automata, formal grammars, and probability.

Human language is a large, artificial construct; the problems involved in natural language processing often depend as much on the structural complexity of language as they do on the computational complexity of the analysis. Perhaps more importantly, language needs to be generated. It is not sufficient for an artificial agent to understand sentences; it must produce them as well.

Natural Language Understanding

The task of Natural Language Understanding can be divided into various subtasks. Following are the key subtasks required for NLU:

- Signal processing
- Syntactic analysis
- Semantic analysis
- Pragmatics

Signal processing is the task of taking a spoken bit of language and turning it into a sequence of words. In many natural language applications, this step can be avoided because the input to the system is already in textual form—consider a machine trying to understand a newspaper article, or trying to translate such an article from one language to another.

Syntax refers to the structure of the sentences being analyzed. The two sentences, "John hit Mary," and "Mary hit John" use the same words. The words mean the same things in each sentence, but the sentences mean different things because the words are in a different order. Syntactic analysis involves parsing sentences to extract whatever information the word order contains.

Semantics involves making sense of the resulting parse by examining the meanings of the words it contains. Some disambiguation may be done here; as an example, "pen" has multiple meanings in English. Semantic analysis selects the appropriate one when distinguishing between the two sentences “The pig is in the pen”. And “The ink is in the pen”.

Finally, **pragmatics** involves understanding the sentence or other text in the context of our overall knowledge base. "Do you know when the train leaves?" is a request for information if asked of a ticket agent, but a simple yes/no question if asked of a friend planning a trip.

Natural Language Generation

Natural language generation (NLG) is the use of artificial intelligence (AI) programming to produce written or spoken narratives from a data set. NLG is related to human-to-machine and machine-to-human interaction, including computational linguistics, natural language processing (NLP) and natural language understanding (NLU).

Research about NLG often focuses on building computer programs that provide data points with context. Sophisticated NLG software can mine large quantities of numerical data, identify patterns and share that information in a way that is easy for humans to understand. The speed of NLG software is especially useful for producing news and other time-sensitive stories on the internet. At its best, NLG output can be published verbatim as web content.

How NLG works

NLG is a multi-stage process, with each step further refining the data being used to produce content with natural-sounding language. The six stages of NLG are as follows:

Content analysis. Data is filtered to determine what should be included in the content produced at the end of the process. This stage includes identifying the main topics in the source document and the relationships between them.

Data understanding. The data is interpreted, patterns are identified and it's put into context. Machine learning is often used at this stage.

Document structuring. A document plan is created and a narrative structure chosen based on the type of data being interpreted.

Sentence aggregation. Relevant sentences or parts of sentences are combined in ways that accurately summarize the topic.

Grammatical structuring. Grammatical rules are applied to generate natural-sounding text. The program deduces the syntactical structure of the sentence. It then uses this information to rewrite the sentence in a grammatically correct manner.

Language presentation. The final output is generated based on a template or format the user or programmer has selected.

References

[1] S. Lucci and D. Kopec, ArtificiAl intelligence in the 21st century: A Living Introduction, Virginia: Mercury Learning and Information, 2016.

[2] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, New Jersey: Pearson Education, Inc., 2010.

[3] E. Rich, K. Knight and S. B. Nair, Artificial Intelligence, Delhi: Tata McGraw-Hill, 2009.

[4] D. L. Poole and A. K. Mackworth, Artificial Intelligence: Foundations of Computational Agents, New York: CAMBRIDGE UNIVERSITY PRESS, 2010.

[5] D. W. Patterson, INTRODUCTION TO ARTIFICIAL INTELLIGENCE AND EXPERT SYSTEMS, New Jersey: Prentice-Hall, Inc, 1990.

[6] A. Kumar, "Adaline Explained with Python Example," Data Analytics, 12 October 2020. [Online]. Available: https://vitalflux.com/adaline-explained-with-python-example/.

[7] "Hebbian Learning," THE DECISION LAB, [Online]. Available: https://thedecisionlab.com/reference-guide/neuroscience/hebbian-learning/.

[8] S. D, "Learning Neural Networks and Learning Rules | Artificial Intelligence," Engineering Notes, [Online]. Available: https://www.engineeringenotes.com/artificial-intelligence-2/neural-network-artificial-intelligence-2/learning-neural-networks-and-learning-rules-artificial-intelligence/35478.

[9] I. Wigmore, "natural language generation (NLG)," SearchEnterpriseAI, [Online]. Available: https://searchenterpriseai.techtarget.com/definition/natural-language-generation-NLG.

[10] A. Singh, "Introduction to Reinforcement Learning," datacamp, 30 November 2018. [Online]. Available: https://www.datacamp.com/community/tutorials/introduction-reinforcement-learning?utm_source=adwords_ppc&utm_medium=cpc&utm_campaignid=14989519638&utm_adgroupid=127836677279&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adpostion=&utm_creativ.