1. INTRODUCTION TO ARTIFICIAL NEURAL NETWORKS

1.1. Artificial intelligence and the machine learning

The term “artificial intelligence” dates back to the mid-1950s, when mathematician John McCarthy, widely recognized as the father of AI, used it to describe machines that do things people might call intelligent. He and Marvin Minsky, whose work was just as influential in the AI field, organized the Dartmouth Summer Research Project on Artificial Intelligence in 1956. A few years later, with McCarthy on the faculty, MIT founded its Artificial Intelligence Project, later the AI Lab. It merged with the Laboratory for Computer Science (LCS) in 2003 and was renamed the Computer Science and Artificial Intelligence Laboratory, or CSAIL.

Now a ubiquitous part of modern society, AI refers to any machine that is able to replicate human cognitive skills, such as problem solving. Over the second half of the 20th century, machine learning emerged as a powerful AI approach that allows computers to, as the name implies, learn from input data without having to be explicitly programmed. Or as formally described by Tom Mitchell (1998), 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.

One technique used in machine learning is a neural network, which draws inspiration from the biology of the brain, relaying information between layers of so-called artificial neurons. The very first artificial neural network was created by Minsky as a graduate student in 1951 (see “Learning Machine, 1951”), but the approach was limited at first, and even Minsky himself soon turned his focus to other approaches for creating intelligent machines. In recent years, neural networks have made a comeback, particularly for a form of machine learning called deep learning, which can use very large, complex neural networks.

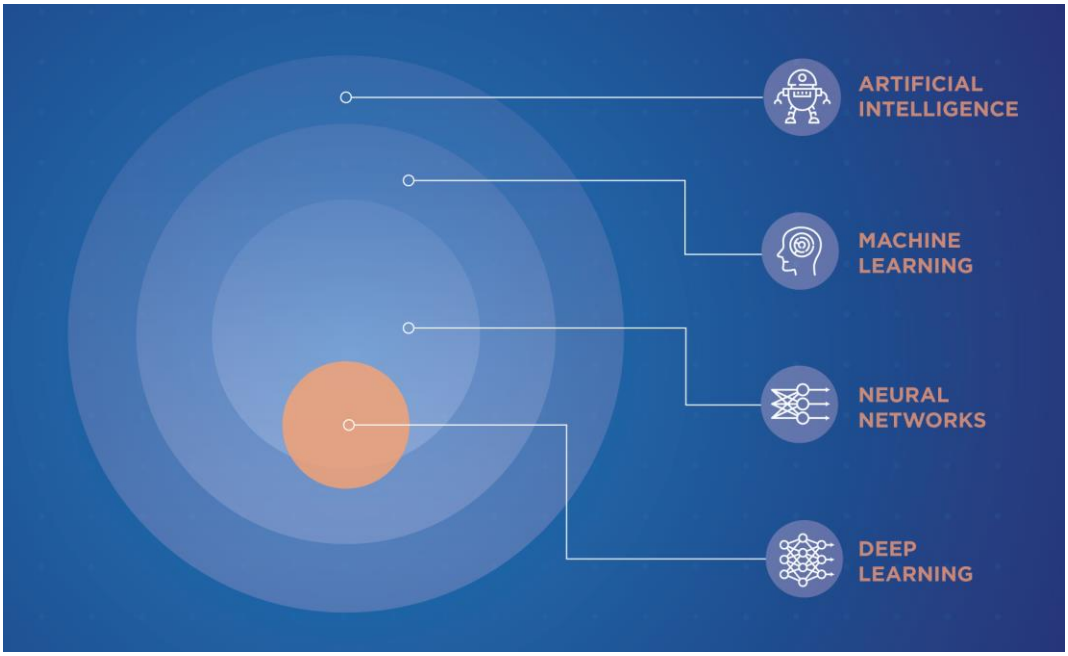


Figure 1: Artificial Intelligence Versus Neural Networks

For the past few years, deep learning and **Artificial Neural Networks** (ANNs) gained a lot of popularity as a machine learning algorithm in a wide variety of fields. These include computer vision, natural language processing/machine translation, speech processing and generation, robotics and self-driving cars. Many tasks which were previously reserved exclusively for humans slowly become automated with ANNs, often with equal or even better performance.

Autopilot & Full Self Driving

In September, we launched Smart Summon in the US which has been used more than one million times to date. This functionality allows car owners to summon their cars from up to 200 feet in a parking lot or driveway. Our neural network learning approach enables us to continue to iterate and improve functionality over time.

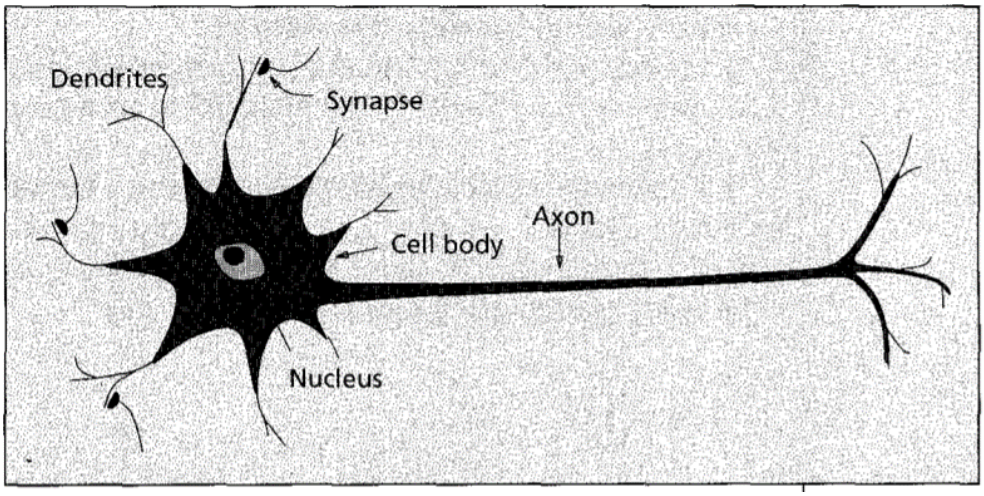
During Q3, we registered one accident for every 4.34 million miles driven in which drivers had Autopilot engaged. This compares to the national average of one accident for every 0.5 million miles based on NHTSA's most recent US data.

Figure 2: safer self-driving car- source: Tesla's Q3 2019 Update

1.2. Biological neural network

A neuron (or nerve cell) is a special biological cell that processes information (see Figure 1). It is composed of a cell body, or soma, and two types of out-reaching tree-like branches: the axon and the dendrites. The cell body has a nucleus that contains information about hereditary traits and a plasma that holds the molecular equipment for producing material needed by the

neuron. A neuron receives signals (impulses) from other neurons through its dendrites (receivers) and transmits signals generated by its cell body along the axon (transmitter), which eventually branches into strands and substrands. At the terminals of these strands are the synapses. A synapse is an elementary structure and functional unit between two neurons (an axon strand of one neuron and a dendrite of another), When the impulse reaches the synapse's terminal, certain chemicals called neurotransmitters are released. The neurotransmitters diffuse across the synaptic gap, to enhance or inhibit, depending on the type of the synapse, the receptor neuron's own tendency to emit electrical impulses. The synapse's effectiveness can be adjusted by the signals passing through it so that the synapses can learn from the activities in which they participate. This dependence on history acts as a memory, which is possibly responsible for human memory.



Neurons are massively connected, much more complex and dense than telephone networks. Each neuron is connected to 10^3 to 10^4 other neurons.

Neurons communicate through a very short train of pulses, typically milliseconds in duration. The message is modulated on the pulse-transmission frequency. This frequency can vary from a few to several hundred hertz, which is a million times slower than the fastest switching speed in electronic circuits. However, complex perceptual decisions such as face recognition are typically made by humans within a few hundred milliseconds. These decisions are made by a network of neurons whose operational speed is only a few milliseconds. This implies that the computations cannot take more than about 100 serial stages. In other words, the brain runs parallel programs that are about 100 steps long for such perceptual tasks. This is known as the hundred step rule. The same timing considerations show that the amount of information sent from one neuron to another must be very small (a few bits). This implies that critical

information is not transmitted directly, but captured and distributed in the interconnections- hence the name, connectionist model, used to describe ANNs.

1.3. Computational models of neurons

McCulloch and Pitts proposed a binary threshold unit as a computational model for an artificial neuron (see Figure 4).

This mathematical neuron computes a weighted sum of its n input signals, $x_j, j = 1,2,\ldots,n$ and generates an output of 1 if this sum is above a certain threshold u. Otherwise, an output of 0 results. Mathematically

$$y = \theta\left(\sum_{j=1}^{n} w_j x_j - u\right),$$

where $\theta(\,)$ is a unit step function at 0, and w_j, is the synapse weight associated with the jth input. For simplicity of notation, we often consider the threshold u as another weight $w_0 = -u$ attached to the neuron with a constant input $x_0 = 1$. Positive weights correspond to excitatory synapses, while negative weights model inhibitory ones. McCulloch and Pitts proved that, in principle, suitably chosen weights let a synchronous arrangement of such neurons perform universal computations. There is a crude analogy here to a biological neuron: wires and interconnections model axons and dendrites, connection weights represent synapses, and the threshold function approximates the activity in a soma. The McCulloch and Pitts model, however, contains a number of simplifying assumptions that do not reflect the true behavior of biological neurons.

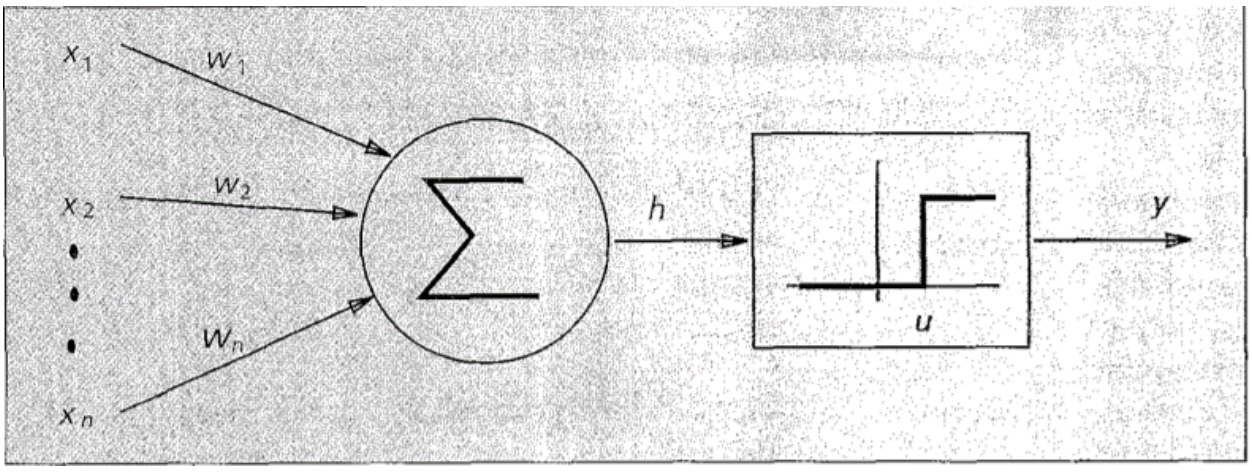


Figure 3: McCulloch-Pitts model of a neuron.

The McCulloch-Pitts neuron has been generalized in many ways. An obvious one is to use activation functions other than the threshold function, such as piecewise linear, sigmoid, or Gaussian, as shown in Figure 4. The two most common activation functions are the sigmoid and hyperbolic tangent activation function (Tanh). sigmoid function is a strictly increasing function that exhibits smoothness and has the desired asymptotic properties. The standard sigmoid function is the logistic function, defined by

$$f(x) = \frac{1}{1 + e^{-x}}$$

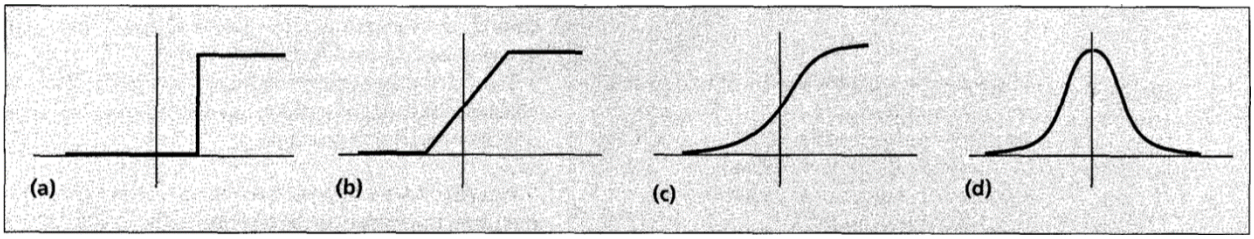


Figure 4: Different types of activation functions: (a) threshold, (b) piecewise linear, (c) sigmoid, and (d) Gaussian.

The hyperbolic tangent activation function is the more common of these two, as has a number range from -1 to 1, compared to the sigmoid function which is only from 0 to 1. The Tanh can be represented mathematically as:

$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

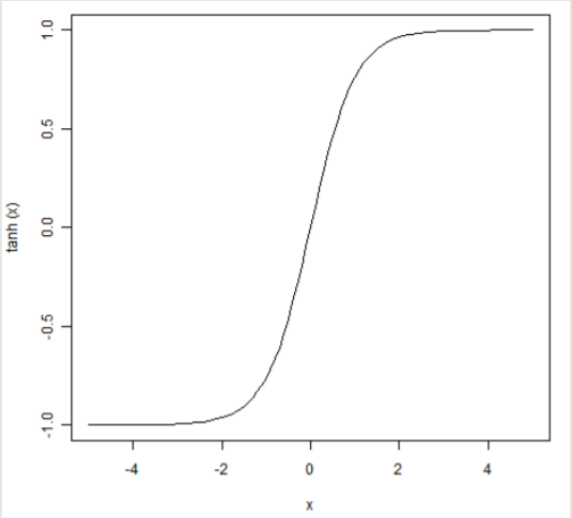


Figure 5: The Hyperbolic Tangent Function.

2. ARTIFICIAL NEURAL NETWORK ARCHITECTURES

ANNs can be viewed as weighted directed graphs in which artificial neurons are nodes and directed edges (with weights) are connections between neuron outputs and neuron inputs.

Based on the connection pattern (architecture), ANNs can be grouped into two categories (see Figure 6):

- feed-forward networks, in which graphs have no loops
- recurrent (or feedback) networks, in which loops occur because of feedback connections.

In the most common family of feed-forward networks, called multilayer perceptron, neurons are organized into layers that have unidirectional connections between them. Figure 6 also shows typical networks for each category.

Different connectivities yield different network behaviors. Generally speaking, feed-forward networks are static, that is, they produce only one set of output values rather than a sequence of values from a given input. Feedforward networks are memory-less in the sense that their response to an input is independent of the previous network state. Recurrent, or feedback, networks, on the other hand, are dynamic systems. When a new input pattern is presented, the neuron outputs are computed. Because of the feedback paths, the inputs to each neuron are then modified, which leads the network to enter a new state.

Different network architectures require appropriate learning algorithms. The next section provides an overview of learning processes.

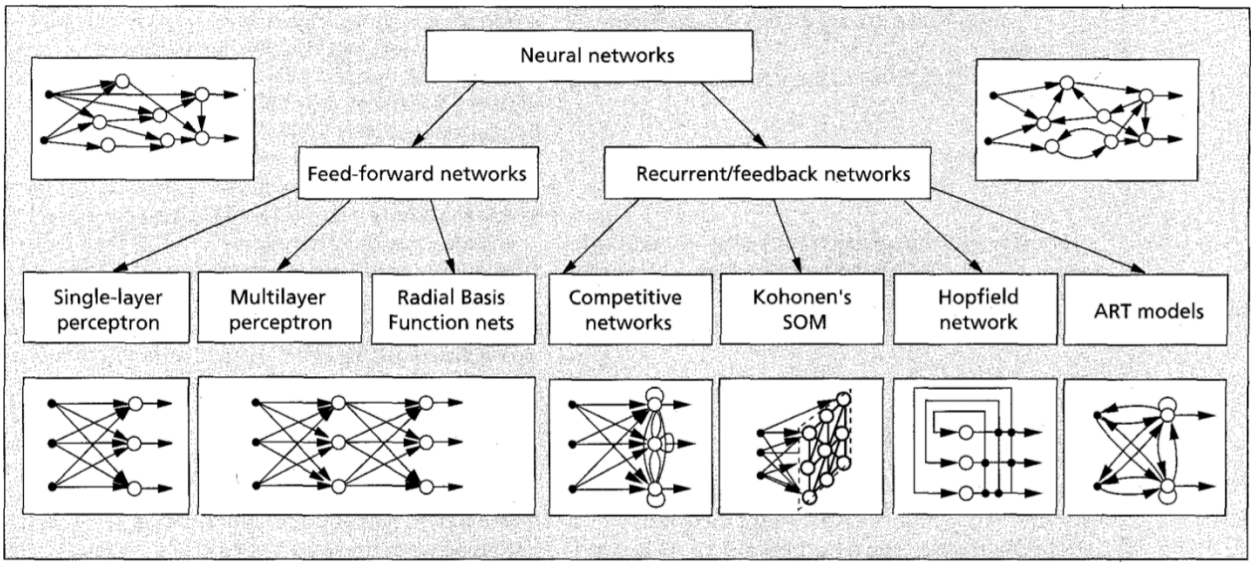


Figure 6: A taxonomy of feed-forward and recurrent/feedback network architectures.

3. NEURAL NETWORK LEARNING

The ability to learn is a fundamental trait of intelligence. Although a precise definition of learning is difficult to formulate, a learning process in the ANN context can be viewed as the problem of updating network architecture and connection weights so that a network can efficiently perform a specific task. The network usually must learn the connection weights from available training patterns. Performance is improved over time by iteratively updating the weights in the network. ANNs' ability to automatically learn from examples makes them attractive and exciting. Instead of following a set of rules specified by human experts, ANNs appear to learn underlying rules (like input-output relationships) from the given collection of representative examples. This is one of the major advantages of neural networks over traditional expert systems.

To understand or design a learning process, you must first have a model of the environment in which a neural network operates, that is, you must know what information is available to the network. We refer to this model as a **learning paradigm**. Second, you must understand how network weights are updated, that is, which learning rules govern the updating process. A **learning algorithm** refers to a procedure in which learning rules are used for adjusting the weights.

There are three main learning paradigms: supervised, unsupervised, and hybrid. In supervised learning, or learning with a “teacher,” the network is provided with a correct answer

(output) for every input pattern. Weights are determined to allow the network to produce answers as close as possible to the known correct answers. Reinforcement learning is a variant of supervised learning in which the network is provided with only a critique on the correctness of network outputs, not the correct answers themselves. In contrast, unsupervised learning, or learning without a teacher, does not require a correct answer associated with each input pattern in the training data set. It explores the underlying structure in the data, or correlations between patterns in the data, and organizes patterns into categories from these correlations. Hybrid learning combines supervised and unsupervised learning. Part of the weights are usually determined through supervised learning, while the others are obtained through unsupervised learning.

4.1. Hebbian learning

The oldest learning rule is Hebb's postulate of learning. Hebb based it on the following observation from neurobiological experiments: If neurons on both sides of a synapse are activated synchronously and repeatedly, the synapse's strength is selectively increased. Mathematically, the Hebbian rule can be described as:

$$w_{ij}(t+1) = w_{ij}(t) + \eta\, y_j(t)\, x_i\,(t)$$

where x_i, and y_j, are the output values of neurons i and j, respectively, which are connected by the synapse w_{ij}, and η is the learning rate. Note that x_i, is the input to the synapse.

An important property of this rule is that learning is done locally, that is, the change in synapse weight depends only on the activities of the two neurons connected by it.

4.2. ERROR-CORRECTION RULES

In the supervised learning paradigm, the network is given a desired output for each input pattern. During the learning process, the actual output y generated by the network may not equal the desired output d. The basic principle of error-correction learning rules is to use the error signal $(d - y)$ to modify the connection weights to gradually reduce this error.

The perceptron learning rule is based on this error-correction principle. A perceptron consists of a single neuron with adjustable weights, $w_j, j = 1,2,\ldots,n$, and threshold u, as shown in Figure 3. Given an input vector $\boldsymbol{x} = (x_1, x_2, , \ldots, x_n)^t$, the net input to the neuron is

$$v = \sum_{j=1}^{n} w_j x_j - u$$

The output y of the perceptron is + 1 if $v > 0$, and 0 otherwise. In a two-class classification problem, the perceptron assigns an input pattern to one class if $y = 1$, and to the other class if $y = 0$.

Note that learning occurs only when the perceptron makes an error. Rosenblatt proved that when training patterns are drawn from two linearly separable classes, the perceptron learning procedure converges after a finite number of iterations. This is the perceptron convergence theorem. In practice, you do not know whether the patterns are linearly separable. Many variations of this learning algorithm have been proposed in the literature. Other activation functions that lead to different learning characteristics can also be used. However, a single-layer perceptron can only separate linearly separable patterns as long as a monotonic activation function is used. The back-propagation learning algorithm is also based on the error-correction principle.

Perceptron learning algorithm

1. Initialize the weights and threshold to small random numbers.
2. Present a pattern vector $(x_1, x_2, \ldots, x_n)^t$ and evaluate the output of the neuron.
3. Update the weights according to

   $$w_j(t+1) = w_j(t) + \eta\,(d - y)\,x_j,$$

   where d is the desired output, t is the iteration number, and η $(0.0 < \eta < 1.0)$ is the gain (step size).

4.3. BOLTZMAN LEARNING

Boltzmann machines are symmetric recurrent networks consisting of binary units (+ 1 for "on" and -1 for "off'). By symmetric, we mean that the weight on the connection from unit i to unit j is equal to the weight on the connection from unit j to unit i ($w_{ij} = w_{ji}$). A subset of the neurons, called visible, interact with the environment; the rest, called hidden, do not.

Each neuron is a stochastic unit that generates an output (or state) according to the Boltzmann distribution of statistical mechanics. Boltzmann machines operate in two modes: clamped, in which visible neurons are clamped onto specific states determined by the environment; and free-running, in which both visible and hidden neurons are allowed to operate freely.

Boltzmann learning is a stochastic learning rule derived from information-theoretic and thermodynamic principles. The objective of Boltzmann learning is to adjust the connection weights so that the states of visible units satisfy a particular desired probability distribution. According to the Boltzmann learning rule, the change in the connection weight w_{ij} is given by

$$\Delta w_{ij} = \eta(\overline{P_{ij}} - P_{ij}),$$

where η is the learning rate, and $\overline{P_{ij}}$, and P_{ij} are the correlations between the states of units i and j when the network operates in the clamped mode and free-running mode, respectively. The values of $\overline{P_{ij}}$, and P_{ij}, are usually estimated from Monte Carlo experiments, which are extremely slow.

Boltzmann learning can be viewed as a special case of error-correction learning in which error is measured not as the direct difference between desired and actual outputs, but as the difference between the correlations among the outputs of two neurons under clamped and free running operating conditions.

4.4. COMPETITIVE LEARNING RULES

Unlike Hebbian learning (in which multiple output units can be fired simultaneously), competitive-learning output units compete among themselves for activation. As a result, only one output unit is active at any given time. This phenomenon is known as winner-take-all. Competitive learning has been found to exist in biological neural network.

Competitive learning often clusters or categorizes the input data. Similar patterns are grouped by the network and represented by a single unit. This grouping is done automatically based on data correlations.

The simplest competitive learning network consists of a single layer of output units as shown in Figure 6. Each output unit i in the network connects to all the input units (x_j's) via weights, $w_{ij}, j = 1,2,\ldots,n$. Each output unit also connects to all other output units via inhibitory weights but has a self-feedback with an excitatory weight. As a result of competition, only the unit i^* with the largest (or the smallest) net input becomes the winner, that is, $\mathbf{W}_i^*.\mathbf{X} > \mathbf{W}_i.\mathbf{X}, \forall i$

or $\|\mathbf{W}_i^* - \mathbf{X}\| \leq \|\mathbf{W}_i - \mathbf{X}\|, \forall i$. When all the weight vectors are normalized, these two inequalities are equivalent.

A simple competitive learning rule can be stated as

$$\Delta w_{ij} = \begin{cases} \eta\left(x_j^u - w_{i*j}\right), & i = i^* \\ 0, & i \neq i^* \end{cases}$$

Note that only the weights of the winner unit get updated. The effect of this learning rule is to move the stored pattern in the winner unit (weights) a little bit closer to the input pattern. Figure 7 demonstrates a geometric interpretation of competitive learning. In this example, we assume that all input vectors have been normalized to have unit length. They are depicted as black dots in Figure 7. The weight vectors of the three units are randomly initialized. Their initial and final positions on the sphere after competitive learning are marked as Xs in Figures 7a and 7b, respectively. In Figure 7, each of the three natural groups (clusters) of patterns has been discovered by an output unit whose weight vector points to the center of gravity of the discovered group.

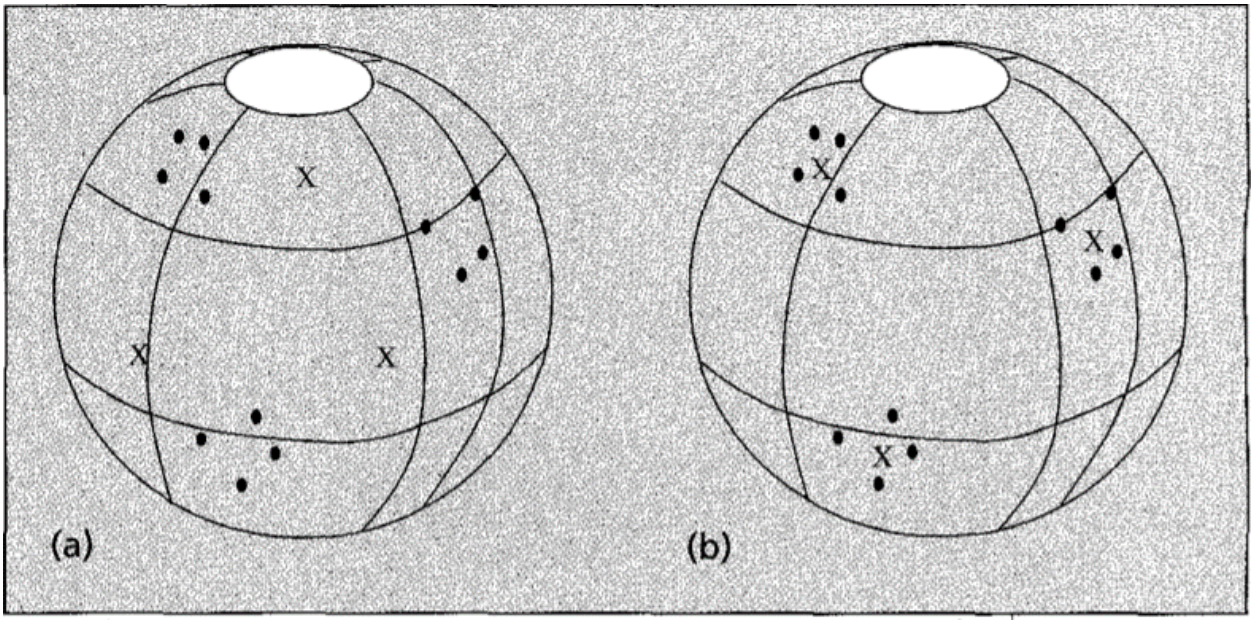


Figure 7: An example of competitive learning: (a) before learning; (b) after learning

You can see from the competitive learning rule that the network will not stop learning (updating weights) unless the learning rate η is 0. A particular input pattern can fire different output units at different iterations during learning. This brings up the stability issue of a learning

system. The system is said to be stable if no pattern in the training data changes its category after a finite number of learning iterations. One way to achieve stability is to force the learning rate to decrease gradually as the learning process proceeds towards 0. However, this artificial freezing of learning causes another problem termed plasticity, which is the ability to adapt to new data. This is known as Grossberg's stability-plasticity dilemma in competitive learning.

The most well-known example of competitive learning is vector quantization for data compression. It has been widely used in speech and image processing for efficient storage, transmission, and modeling. Its goal is to represent a set or distribution of input vectors with a relatively small number of prototype vectors (weight vectors), or a codebook. Once a codebook has been constructed and agreed upon by both the transmitter and the receiver, you need only transmit or store the index of the corresponding prototype to the input vector. Given an input vector, its corresponding prototype can be found by searching for the nearest prototype in the codebook.

References and further reading:

The Scientist Magazine, A Primer: Artificial Intelligence Versus Neural Networks. https://www.the-scientist.com/magazine-issue/artificial-intelligence-versus-neural-networks-65802

Jain, A. K., Mao, J., & Mohiuddin, K. M. (1996). Artificial neural networks: A tutorial. Computer, 29(3), 31-44.