Lecture Course in

Intelligent Control Systems

4th Class in the Control and Systems

Engineering Department

University of Technology

CCE-CN432

Edited By:

Dr. Mohammed Y. Hassan, Ph. D.

Syllabus

Fourth Year. **Theoretical: 2 Hr./ Week**

CCE-CN432 **(Semester)**

1. Introduction: **(2 Hrs)**

2. Neural Networks (NNs) **(6 Hrs)**

- **Artificial Neuron, Types of Non-Linearities, Why Sigmoid.**
- **Types of NNs, Feed-forward, Feedback, Supervised and Unsupervised.**
- **Learning Algorithms, Basic Delta Rule, Back propagation, Associative memory.**
- **Architectures, Hopfield, Hamming, Kohonen, Neocognition.**
- **Applications.**

3. Fuzzy Logic (FL):

- **Introduction, Fuzzy Concepts, Fuzzy Sets, Fuzzy operations, Fuzzification (Types of).** **(6 Hrs)**
- **Inference Engine, Rule-Base, Types of Defuzzification, Fuzzy Logic Control (FLC).** **(6 Hrs)**

4. Genetic Algorithms (Gas):

- **Introduction, Elements of Gas, Genetic Operators, Initialization, Coding.** **(4 Hrs)**
- **Fitness Function, Selection, Crossover, Mutation.** **(4 Hrs)**

5. Hybrid Systems **(2 Hrs)**

Total: 30 Hrs

References:

1. "Introduction to Artificial Neural Systems"
   By: Jacek M. Zurada, 1999.
2. "Fuzzy Control"
   By: Kevin M. Passino and Stephen Yurkovich, 1998.
3. "Practical Genetic Algorithms"
   By: Randy L. Haupt and Sue Ellen Haupt, 2004

1. Introduction:

Over many centuries, tools of increasing sophistication have been developed to serve the human race. Physical tools such as chisels, hammers, spears, arrows, guns, carts, cars, and aircraft all have their place in the history of civilization. The human race has also developed tools of communication — spoken language, written language, and the language of mathematics. These tools have not only enabled the exchange and storage of information, but have also allowed the expression of concepts that simply could not exist outside of the language.

The last few decades have seen the arrival of a new tool — the digital computer. Computers are able to perform the same sort of numerical and symbolic manipulations that an ordinary person can, but faster and more reliably. They have therefore been able to remove the tedium from many tasks that were previously performed manually, and have allowed the achievement of new feats. Such feats range from huge scientific "number-crunching" experiments to the more familiar electronic banking facilities.

Although these uses of the computer are impressive, it is actually only performing quite simple operations, albeit rapidly. In such applications, the computer is still only a complex calculating machine. The intriguing idea now is whether we can build a computer (or a computer program) that can *think*.

Research in artificial intelligence (or simply AI) is directed toward building such a machine and improving our understanding of intelligence. The ultimate achievement in this field would be to construct a machine that can mimic or exceed human mental capabilities, including reasoning, understanding, imagination, recognition, creativity, and emotions. For instance, machines are now able to play chess at the highest level, to interpret spoken sentences, and to diagnose medical complaints. An objection to these claimed successes might be that the machine does not tackle these problems in the same way that a human would.

In achieving these modest successes, research into artificial intelligence, together with other branches of computer science, has resulted in the development of several useful computing tools.

These tools have a range of potential applications their use in engineering and science. The tools of particular interest can be roughly divided among knowledge-based systems, computational intelligence, and hybrid systems.

Knowledge-based systems include expert and rule-based systems, object-oriented and frame-based systems, and intelligent agents. Computational intelligence includes neural networks, genetic algorithms and other optimization algorithms. Techniques for handling uncertainty, such as fuzzy logic, fit into both categories.

Knowledge-based systems, computational intelligence, and their hybrids are collectively referred to here as *intelligent systems*.

Knowledge-based systems

The principal difference between a knowledge-based system (KBS) and a conventional program lies in the structure. In a conventional program, domain knowledge is intimately intertwined with software for controlling the application of that knowledge. In a knowledge-based system, the two roles are explicitly separated. In the simplest case there are two modules — the knowledge module is called the *knowledge base*, and the control module is called the *inference engine* . In more complex systems, the inference engine itself may be a knowledge-based system containing *meta-knowledge*, i.e., knowledge of how to apply the domain knowledge.

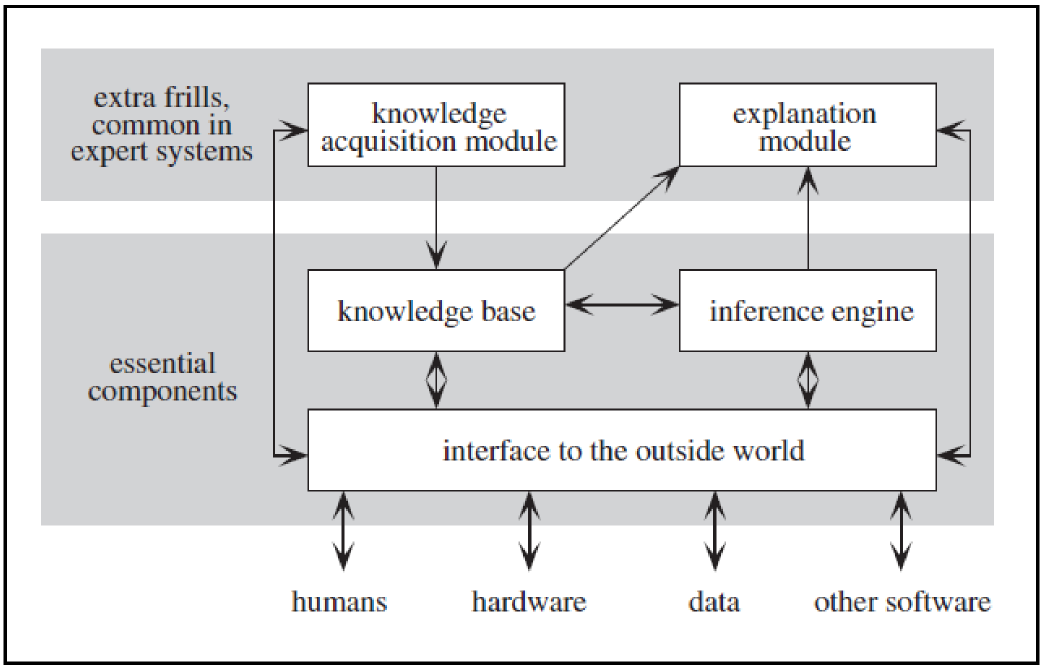


Figure . The main components of a knowledge-based system

The explicit separation of knowledge from control makes it easier to add new knowledge, either during program development or in the light of experience during the program's lifetime. There is an analogy with the brain, the control processes of which are approximately unchanging in their nature (cf. the inference engine), even though individual behavior is continually modified by new knowledge and experience (cf. updating the knowledge base).

Expert systems

Expert systems are a type of knowledge-based system designed to embody expertise in a particular specialized domain. Example domains might be configuring computer networks, diagnosing faults in telephones, or mineral prospecting. An expert system is intended to act as a human expert who can be consulted on a range of problems that fall within his or her domain of expertise. Typically, the user of an expert system will enter into a dialogue in which he or she describes the problem (such as the symptoms of a fault) and the expert system offers advice, suggestions, or recommendations. The dialogue may be led by the expert system, so that the user responds to a series of questions or enters information into a spreadsheet.

Since an expert system is a knowledge-based system that acts as a specialist consultant, it is often proposed that an expert system must offer certain capabilities that mirror those of a human consultant. In particular, it is often claimed that an expert system must be capable of justifying its current line of inquiry and explaining its reasoning in arriving at a conclusion.

Computational intelligence

The knowledge-based systems are all symbolic representations, in which knowledge is explicitly represented in words and symbols that are combined to form rules, facts, relations, or other forms of knowledge representation. As the knowledge is explicitly written, it can be read and understood by a human. These symbolic techniques contrast with numerical techniques such as genetic algorithms and neural networks.

Here the knowledge is not explicitly stated but is represented by numbers which are adjusted as the system improves its accuracy.

These techniques are collectively known as *computational intelligence* (CI) or *soft computing*.

soft computing is defined as "a collection of computational techniques in computer science, AI, machine learning and some engineering disciplines, which attempt to study, model, and analyze very complex phenomena: those for which more conventional methods have not yielded low cost, analytic, and complete solutions." The typical techniques that belong to the soft computing arena include artificial neural networks (ANNs), fuzzy sets and systems, evolutionary computation including evolutionary strategies (ESs), swarm intelligence and harmony search, Bayesian network, chaos theory, etc. Much of these soft computing techniques are inspired by biological processes or are the results of attempts to emulate such processes.

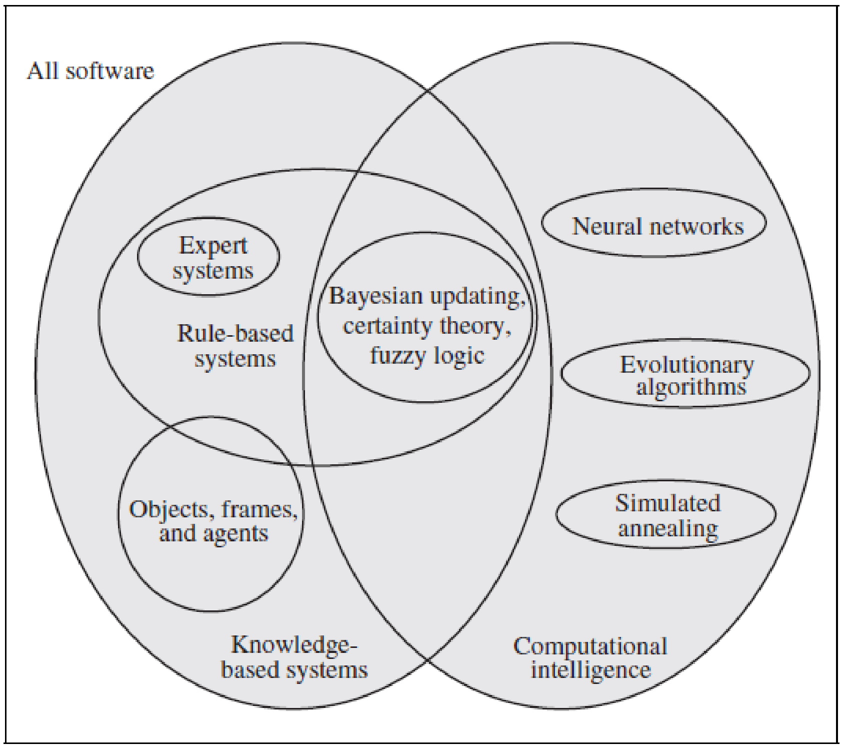


Figure Categories of intelligent system software

- neural networks;
- genetic algorithms or, more generally, evolutionary algorithms;
- probabilistic methods such as Bayesian updating and certainty factors;
- fuzzy logic;
- combinations of these techniques with each other and with KBSs.

Examples of systems that would require and benefit from intelligent control and optimization include electric power plants and their distributed networks, military command and control systems, air traffic control systems, biological systems, and manufacturing systems.

2. Neural Network (NN)

Introduction:

Computers are extremely fast and precise in executing sequences of instructions that have been formulated for them. A human information processing system is composed of neurons switching at a speed about a million times slower than computer gates. In the contrary, humans are more efficient than computers at computationally complex tasks such as speech and pattern recognition.

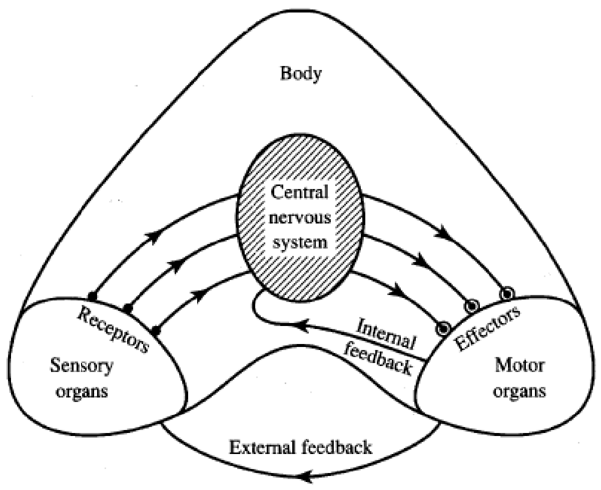


Figure Information flow in nervous system.

Artificial neural networks:

Artificial neural networks (or it can also be called neural networks) are “cellular systems which acquire, store, and utilize experimental knowledge. The knowledge is in the form of stable states or mapping embedded in the networks that can be recalled by the user”.

It has the ability to reproduce some of the flexibility and power of the human brain by artificial means. Neural networks must be taught or trained to learn new associations and new patterns. Learning corresponds to parameter changes. Learning rules and algorithms used for experimental training of networks replace the programming required for conventional computation.

Neural network users do not specify an algorithm to be executed by each computing node as would programmers of a more traditional machine. Instead, they select what in their view is the best architecture, specify the characteristics of the neurons and initial weights, and choose the training mode for the network. Appropriate inputs are then applied to the network so that it can acquire knowledge from the environment. As a result of such exposure, the network assimilates the information that can later be recalled by the user. Network computation is performed by a dense mesh of computing nodes and connections. They operate collectively and simultaneously on most or all data and inputs.

Biological Neurons:

A human brain consists of approximately 10" computing elements called neurons. They communicate through a connection network of axons and synapses having a density of approximately 10^4 synapses per neuron.

- Neurons communicate with each other by means of electrical impulses**.** The neurons operate in a chemical environment that is even more important in terms of actual brain behaviour.
- The input to the network is provided by sensory receptors. Receptors deliver stimuli both from within the body, as well as from sense organs when the stimuli originate in the external world.
- As a result of information processing in the central nervous systems, the effectors are controlled and give human responses in the form of diverse actions.
- The elementary nerve cell, called a neuron, is the fundamental building block of the biological neural network.

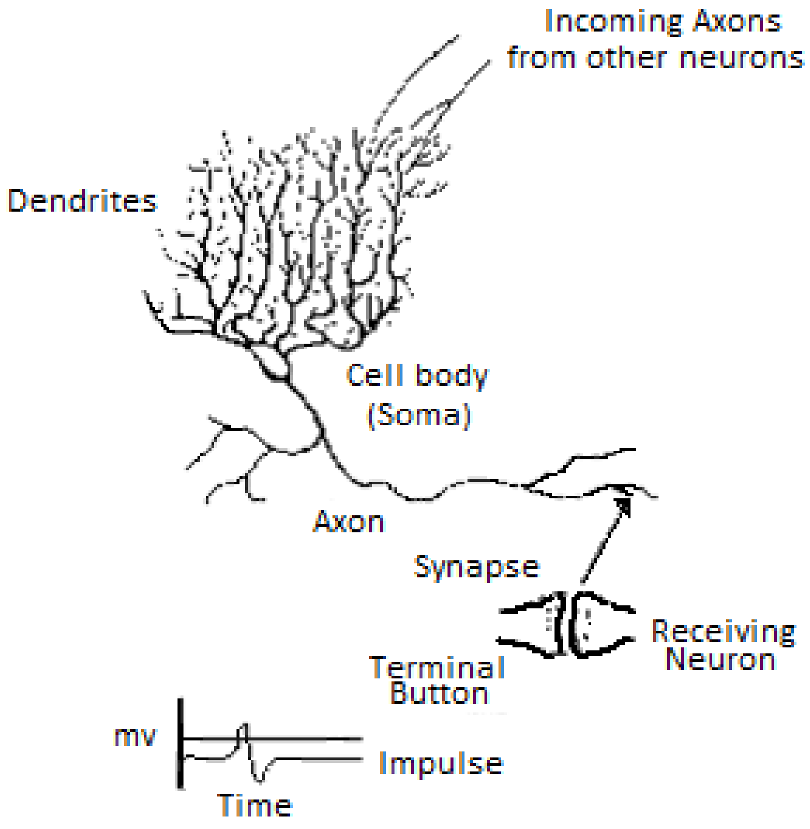


Figure. Structure of a neuron.

- A typical cell has three major regions: the cell body, which is also called the soma, the axon**,** and the dendrites.
- Dendrites form a dendrites tree, which is a very fine bush of thin fibbers around the neuron's body. They receive information from neurons through axons-long fibres that serve as transmission lines.
- An axon is a long cylindrical connection that carries impulses from the

neuron. The end part of an axon splits into a fine arborisation. Each branch of it terminates in a small end bulb almost touching the dendrites of neighbouring neurons.

  - The axon-dendrite contact organ is called a synapse. The synapse is where the neuron introduces its signal to the neighbouring neuron. The signals reaching a synapse and received by dendrites are electrical impulses.
  - The inter-neuronal transmission is sometimes electrical but is usually affected by the release of chemical transmitters at the synapse. Thus, terminal buttons generate the chemical that affects the receiving neuron.
  - The receiving neuron either generates an impulse to its axon, or produces no response.

- The neuron is able to respond to the total of its inputs aggregated within a short time interval called the period of latent summation.
- The neuron's response is generated if the total potential of its membrane reaches a certain level. Specifically, the neuron generates a pulse response and sends it to its axon only if the conditions necessary for firing are fulfilled.
- Incoming impulses can be *excitatory* if they cause the firing, or *inhibitory* if they hinder the firing of the response. A more precise condition for firing is that the excitation should exceed the inhibition by the amount called the threshold of the neuron, typically a value of about 40mV
- Since a synaptic connection causes the excitatory or inhibitory reactions of the receiving neuron, it is practical to assign positive and negative unity weight values, respectively, to such connections.
- This allows us to reformulate the neuron's firing condition. The neuron fires when the total of the weights to receive impulses exceeds the threshold value during the latent summation period.
- After carrying a pulse, an axon fibre is in a state of complete non-excitability for a certain time called the refractory period. For this time interval the nerve does not conduct any signals, regardless of the intensity of excitation.
- Thus, we may divide the time scale into consecutive intervals, each equal to the length of the refractory period. This will enable a discrete-time description of the neurons' performance in terms of their states at discrete time instances.

Applications of Neural Network:

Neural networks are useful and applicable for solving real world problems, like:

1. Pattern recognition. (Like Image processing, speech recognition, robotics...etc)
2. Risk assessment.
3. Diagnostics (in medicine, engineering, manufacturing ... etc).
4. Data compression.
5. Optimization (Like sales man route).
6. Addressable memories.

7. Control.
8. Weather Forecasting.

Elements and modeling of neural networks:

The basic processing elements of Neural Network are called neurons or nodes. Neurons perform as summing and nonlinear mapping junctions. In some cases they can be considered as threshold units that fire when their total input exceeds certain bias levels. They are often organized in layers and feedback connections both within layer and towards adjacent layers are also allowed. The strength of each connection is expressed by a numerical value called a weight, which can also be modified. Neurons usually operate in parallel and are configured in regular architectures.

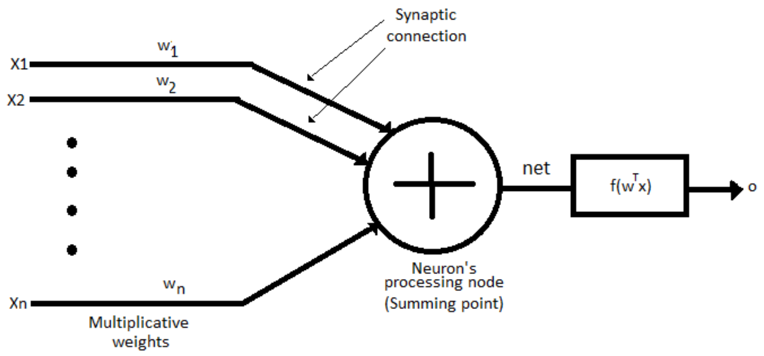


Figure. General symbol of a neuron.

The signal flow of neuron inputs, Xi, is considered to be unidirectional as indicated by arrows, as is a neuron's output signal flow. This symbolic representation, Figure above, shows a set of weights and the *neuron's processing unit,* or *node.* The neuron output signal is given by the following relationship:

$o = f(w^T.X),$

or

$o = f(\sum_{i=1}^{n} w_i.X_i),$

where w is the weight vector defined as:

$w \triangleq [w_1 \quad w_2 \ldots\ldots w_n]^T$

and

$X \triangleq [X_1 \quad X_2 \ldots\ldots X_n]^T$

x is the input vector.

The function $f(w^T.X)$ is often referred to as an *activation function.* Its domain is the set of activation values, *net,* of the neuron model, it is often written as f*(net).* The variable *net* is defined as a scalar product of the weight and input vector:

$net \triangleq w^T.X$

Types of Non-Linearities (Activation Functions):

Activation functions can be divided into two categories:

1) Soft-limiting (continuous) activation functions:

Typical of these activation functions used are:

a) bipolar sigmoid:

$f(net) \triangleq \frac{2}{1+e^{-\lambda net}}$-1,

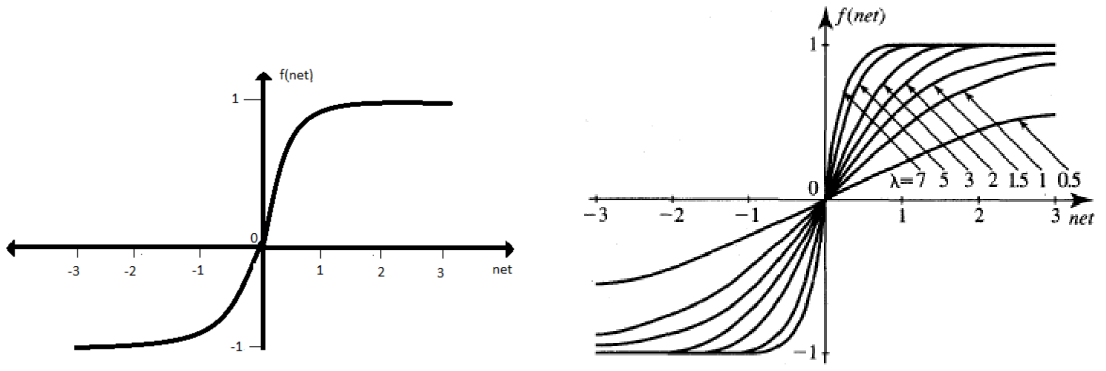


Figure. Bipolar Sigmoid activation function.

where λ > 0 is proportional to the neuron gain determining the steepness of the continuous function f(net) near net=0.

b) Unipolar sigmoid:
By shifting and scaling the bipolar activation function, a unipolar continuous activation function can be obtained:

$$f(net) \triangleq \frac{1}{1+e^{-\lambda net}}$$

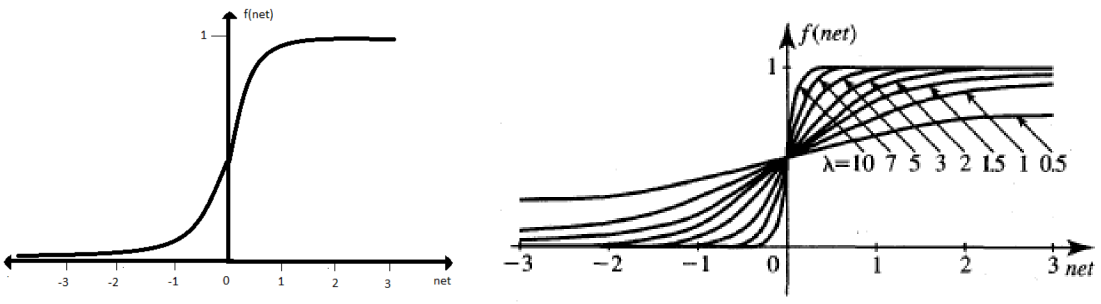


Figure. Unipolar Sigmoid activation function.

c) Linear:
A linear function can also be used as an activation function:
$f(net) = net.$

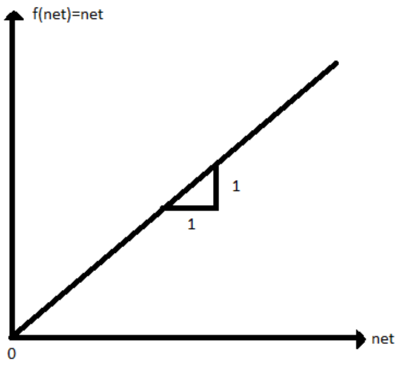


Figure. Example of a linear activation function.

2) Hard-limiting (binary) activation functions:

a) Bipolar binary:

As $\lambda \to \infty$ in the bipolar continuous activation function, the bipolar binary activation function is obtained:

$$f(net) \triangleq sgn(net) = \begin{cases} +1 & net \geq 0 \\ -1 & net < 0 \end{cases}$$

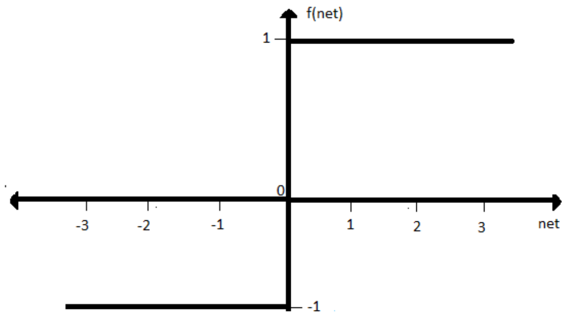


Figure. Example of a Bipolar binary activation function.

b) Unipolar binary:

As $\lambda \to \infty$ in the unipolar continuous activation function, the bipolar binary activation function is obtained:

$$f(net) \triangleq sgn(net) = \begin{cases} +1 & net \geq 0 \\ 0 & net < 0 \end{cases}$$

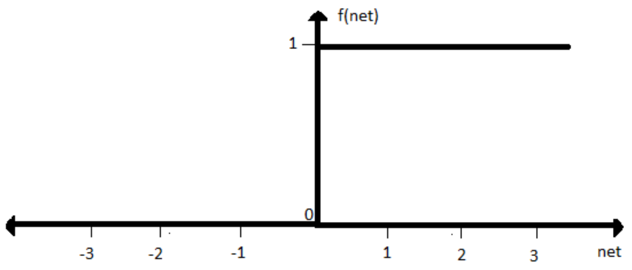


Figure. Example of a unipolar binary activation function.

Note:

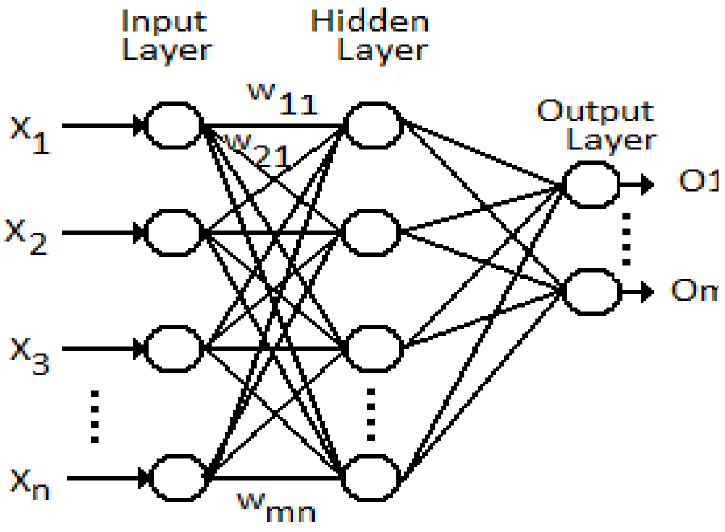
Any function, f*(net)*, that is monotonically *increasing and continuous* such that $net \in R$ *and* $f(net) \in (-1, 1)$ can be used instead.

Types of Neural Networks (NNs):

Neural networks can be defined as an interconnection of neurons such that neurons outputs are connected through weights to all other neurons including themselves; both lag-free and delay connections are allowed. Therefore; one can define two general types of artificial neural networks:

a) Feed-forward and feedback networks:

The mapping of an input pattern into an output pattern in the feed-forward neural network is of feed-forward and instantaneous type, since it involves no time delay between the input and output.



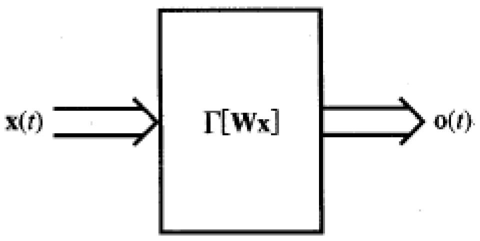
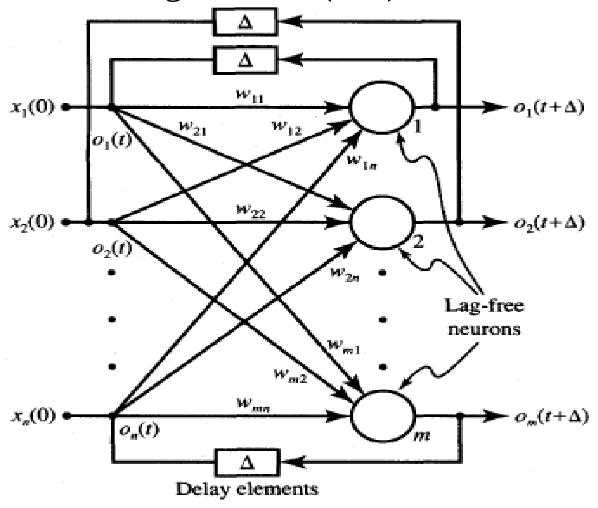


Figure. Muti-layer Feed-forward neural network.

b) Feedback neural network:

It is obtained from the feed-forward one by connecting the neuron’s output to their inputs. In this case, the present output, o(t), controls the output at the following instant o(t+Δ). This network is also called Recurrent neural network.



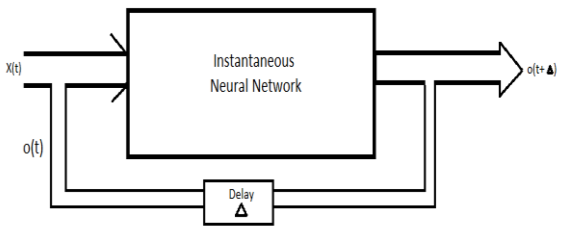


Figure. Feedback neural network.

Neural network Recall:

The process of computation of the output response (o) for a given input (x) performed by the network is known as *recall.*

- Recall is the proper processing phase for a neural network and its objective is to retrieve the information.
- Recall corresponds to the decoding of the stored content which may have been encoded in a network previously.

Types of recall patterns:

- Autoassociation.
  If a set of patterns can be stored, then if the network is presented with a pattern similar to a member of the stored set, it may associate the input with the closest stored pattern. The process is called autoassociation.

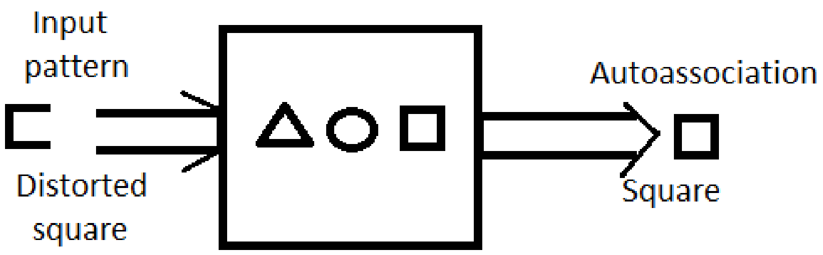


Figure (10) Autoassociation.

- Hetroassociation:
  In hetroassociation processing, the association between pairs of patterns are stored.

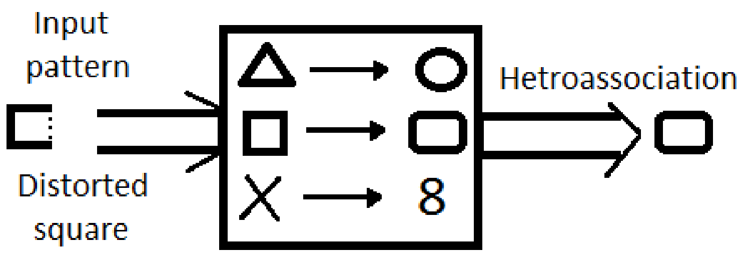


Figure (11) Hetroassociation.

- Classification:
  If a set of input patterns is divided into a number of class or categories, then in response to an input pattern, the classifier should recall the information regarding class membership of the input pattern. Typically, classes are expressed by discrete-values output vectors, and thus output neurons of classifiers would employ binary activation functions.

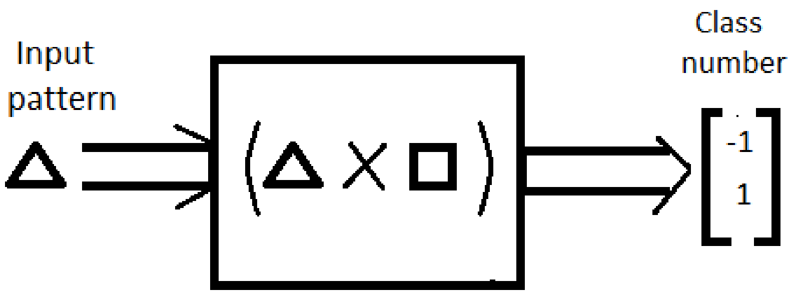
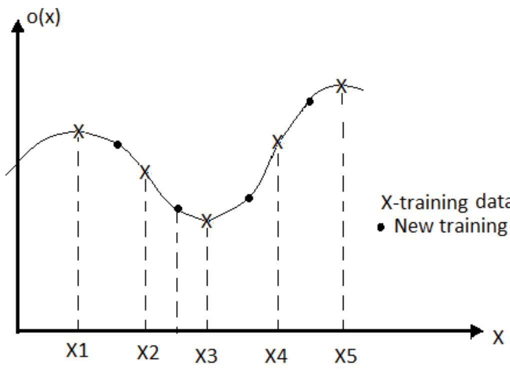


Figure (12) Classification .

- Generalization:

One of the distinct strengths of neural networks is their ability to generalize. The network is said to generalize well when it sensibly interpolates input patterns that are new to the network.



Figure(13) Generalization.

Supervised and unsupervised Learning modes:

Learning is necessary when the information about inputs/outputs is unknown or incomplete, so that no design of a network can be performed in advance. The majority of the neural networks covered in this text require training in supervised or unsupervised learning modes:

- Supervised learning mode:

  In this type of learning we assume that at each instant of time when the input (X) is applied, the desired response (d) (teacher or supervisor) of the input is provided. The distance between the actual and desired response serves as an error measure and is used to correct network parameters externally (weights and biases) so that the error decreases. Since we assume adjustable weights, the teacher may implement a reward-and-punishment scheme to adapt the network's weight matrix (W).

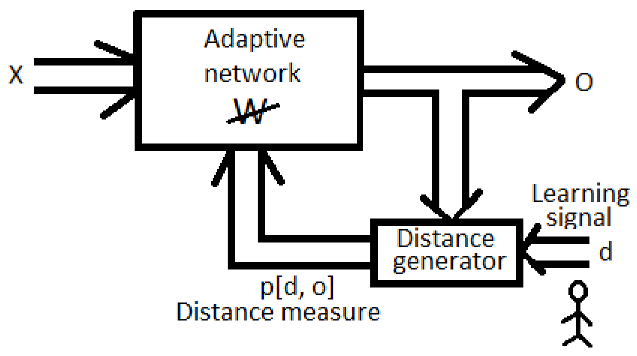


Figure (14) Supervised learning algorithm.

- Unsupervised learning mode (learning without teacher):
  In this mode of learning, the desired response is not known; thus, explicit error information cannot be used to improve network behaviour. Since no information is available as to correctness or incorrectness of responses, learning must be somehow be accomplished based on observations of responses to inputs that we have marginal or no knowledge about.

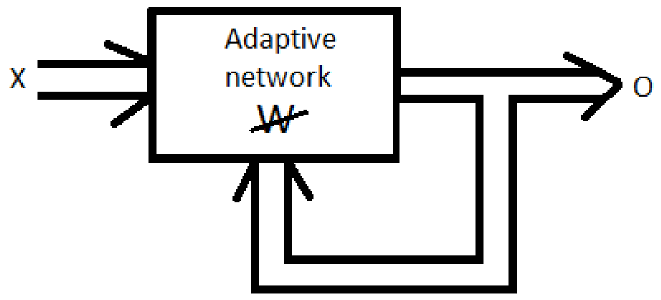


Figure (15) Unsupervised learning algorithm.

Unsupervised learning algorithms use patterns that are typically redundant raw data having no labels regarding their class membership, or associations. In this mode of learning, the network must discover for itself any possibly existing patterns, regularities, separating properties, ...etc. while discovering these, the network undergoes change of its parameters which is called self-organization.

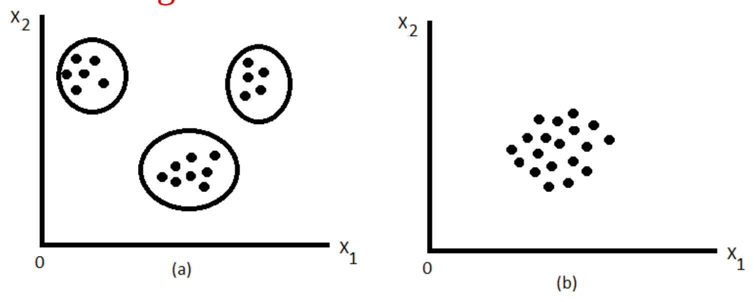


Figure (16) Two-dimensional patterns: a) Clustered b) No apparent clusters

- Learning with supervision corresponds to classroom learning with the teacher’s questions answered by students and corrected, if needed, by the teacher.

- Learning without supervision corresponds to leaning the subject from videotape lecture covering the material but not including any other teacher's involvement. Therefore, the student cannot get explanations of unclear questions, check answers and become fully informed.

1. Learning rules:

A neuron is considered to be an adaptive element. Its weights are modified depending on the input signal, its output value, and the associated teacher response. In some cases, the teacher signal is not available and no error information can be used (unsupervised learning method).
For the neural network shown below, the jth input can be an output of another neuron or it can be an external input. Under different learning rules, the form of the neuron's activation function may be different. Note that the threshold parameter may be included in learning as one of the weights. This would require fixing one of the inputs, say X_n, (Ex.: -1).

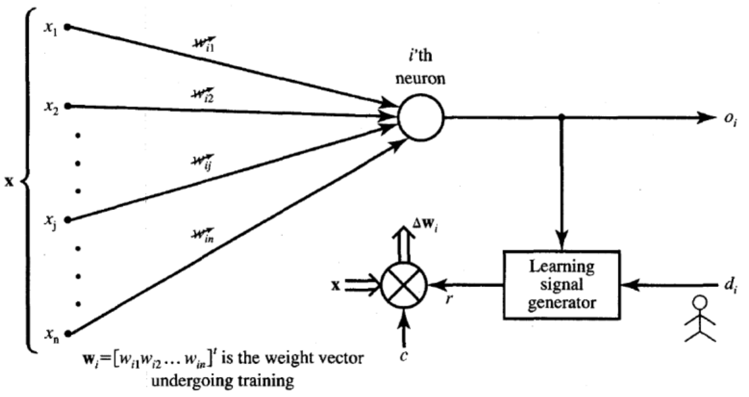


Figure (17) Single neuron

The following *general learning rule* is adopted in neural network studies (Amari 1990): *The weight vector* $\mathbf{w}_i = [w_{i1} \quad w_{i2} \quad \cdots \quad w_{in}]^t$ *increases in proportion to the product of input* $\mathbf{x}$ *and learning signal* r. The learning signal r is in general a function of $\mathbf{w}_i, \mathbf{x}$, and sometimes of the teacher's signal d_i. We thus have for the network shown in Figure(17):

$$r = r(\mathbf{w}_i, \mathbf{x}, d_i)$$

The increment of the weight vector $\mathbf{w}_i$ produced by the learning step at time t according to the general learning rule is:

$$\Delta \mathbf{w}_i(t) = cr\left[\mathbf{w}_i(t), \mathbf{x}(t), d_i(t)\right] \mathbf{x}(t)$$

where c is a positive number called the *learning constant* that determines the rate of learning. The weight vector adapted at time t becomes at the next instant, or learning step,

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + cr\left[\mathbf{w}_i(t), \mathbf{x}(t), d_i(t)\right] \mathbf{x}(t)$$

The superscript convention will be used in this text to index the discrete-time training steps. For the k'th step:

$$\mathbf{w}_i^{k+1} = \mathbf{w}_i^k + cr(\mathbf{w}_i^k, \mathbf{x}^k, d_i^k)\mathbf{x}^k$$

The learning above assumes the form of a sequence of discrete-time weight modifications. Continuous-time learning can be expressed as

$$\frac{\mathrm{d}\mathbf{w}_i(t)}{\mathrm{d}t} = cr\mathbf{x}(t)$$

There are several methods used to train the neural network, in the following is the explanation of two methods; i.e Hebbian learning rule (unsupervised learning) and Perceptron learning rule (supervised learning).

Hebbian Learning Rule

For the Hebbian learning rule the learning signal is equal simply to the neuron's output (Hebb 1949). We have

$$r \overset{\Delta}{=} f(\mathbf{w}_i^t \mathbf{x})$$

The increment $\Delta\mathbf{w}_i$ of the weight vector becomes

$$\Delta\mathbf{w}_i = cf(\mathbf{w}_i^t \mathbf{x})\mathbf{x}$$

The single weight w_{ij} is adapted using the following increment:

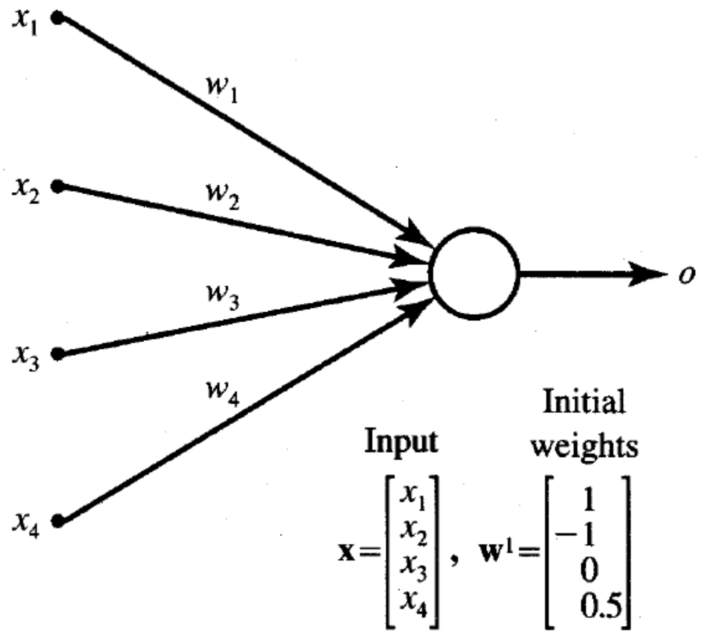
$$\Delta w_{ij} = cf(\mathbf{w}_i^t \mathbf{x})x_j$$

This can be written briefly as

$$\Delta w_{ij} = co_i x_j, \quad \text{for } j = 1, 2, \dots, n$$

This learning rule requires the weight initialization at small random values around $\mathbf{w}_i = \mathbf{0}$ prior to learning. The Hebbian learning rule represents a purely feedforward, unsupervised learning.

Example 1:
This example illustrates Hebbian learning with binary and continuous activation functions of a very simple network. Assume the network shown in Figure below:


x_1
x_2
x_3
x_4
w_1
w_2
w_3
w_4
o
Input
Initial weights
$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$, $\mathbf{w}^1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}$

With the initial weight vector:

$$\mathbf{w}^1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}$$

needs to be trained using the set of three input vectors as below

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix}, \quad \mathbf{x}_2 = \begin{bmatrix} 1 \\ -0.5 \\ -2 \\ -1.5 \end{bmatrix}, \quad \mathbf{x}_3 = \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1.5 \end{bmatrix}$$

for an arbitrary choice of learning constant $c = 1$. Since the initial weights are of nonzero value, the network has apparently been trained beforehand. Assume first that bipolar binary neurons are used, and thus $f(net) = \operatorname{sgn}(net)$.

Step 1 Input $\mathbf{x}_1$ applied to the network results in activation net^1 as below:

$$net^1 = \mathbf{w}^{1t}\mathbf{x}_1 = \begin{bmatrix} 1 & -1 & 0 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix} = 3$$

The updated weights are

$$\mathbf{w}^2 = \mathbf{w}^1 + \operatorname{sgn}(net^1)\mathbf{x}_1 = \mathbf{w}^1 + \mathbf{x}_1$$

and plugging numerical values we obtain

$$\mathbf{w}^2 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} + \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ -3 \\ 1.5 \\ 0.5 \end{bmatrix}$$

where the superscript on the right side of the expression denotes the number of the current adjustment step.

Step 2 This learning step is with $\mathbf{x}_2$ as input:

$$net^2 = \mathbf{w}^{2t}\mathbf{x}_2 = \begin{bmatrix} 2 & -3 & 1.5 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ -0.5 \\ -2 \\ -1.5 \end{bmatrix} = -0.25$$

The updated weights are

$$net^3 = \mathbf{w}^{3t}\mathbf{x}_3 = \begin{bmatrix} 1 & -2.5 & 3.5 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1.5 \end{bmatrix} = -3$$

The updated weights are

$$\mathbf{w}^4 = \mathbf{w}^3 + \text{sgn}\,(net^3)\mathbf{x}_3 = \mathbf{w}^3 - \mathbf{x}_3 = \begin{bmatrix} 1 \\ -3.5 \\ 4.5 \\ 0.5 \end{bmatrix}$$

It can be seen that learning with discrete $f(net)$ and $c = 1$ results in adding or subtracting the entire input pattern vectors to and from the weight vector, respectively. In the case of a continuous $f(net)$, the weight incrementing/decrementing vector is scaled down to a fractional value of the input pattern.

Revisiting the Hebbian learning example, with continuous bipolar activation function $f(net)$, using input $\mathbf{x}_1$ and initial weights $\mathbf{w}^1$, we obtain neuron output values and the updated weights for $\lambda = 1$ as summarized in Step 1. The only difference compared with the previous case is that instead of $f(net) = \text{sgn}\,(net)$, now the neuron's response is computed **using bipolar continuous activation function**

Step 1

$$f(net^1) = 0.905$$

$$\mathbf{w}^2 = \begin{bmatrix} 1.905 \\ -2.81 \\ 1.357 \\ 0.5 \end{bmatrix}$$

Subsequent training steps result in weight vector adjustment as below:

Step 2

$$f(net^2) = -0.077$$

$$\mathbf{w}^3 = \begin{bmatrix} 1.828 \\ -2.772 \\ 1.512 \\ 0.616 \end{bmatrix}$$

Step 3

$$f(net^3) = -0.932$$

$$\mathbf{w}^4 = \begin{bmatrix} 1.828 \\ -3.70 \\ 2.44 \\ -0.783 \end{bmatrix}$$

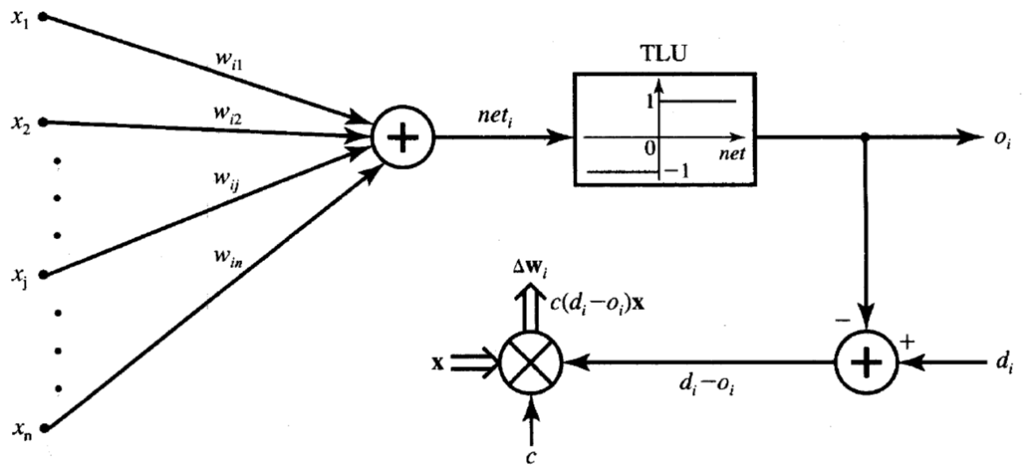
Comparison of learning using discrete and continuous activation functions indicates that the weight adjustments are tapered for continuous $f(net)$ but are generally in the same direction.

Perceptron Learning Rule

For the perceptron learning rule, the learning signal is the difference between the desired and actual neuron's response (Rosenblatt 1958). Thus, learning is supervised and the learning signal is equal to

$$r \triangleq d_i - o_i$$

where $o_i = \operatorname{sgn}(\mathbf{w}_i^t\mathbf{x})$, and d_i is the desired response as shown in Figure **below:**



Weight adjustments in this method, $\Delta \mathbf{w}_i$ and Δw_{ij}, are obtained as follows

$$\Delta \mathbf{w}_i = c\,[d_i - \operatorname{sgn}(\mathbf{w}_i^t \mathbf{x})]\,\mathbf{x}$$

$$\Delta w_{ij} = c\,[d_i - \operatorname{sgn}(\mathbf{w}_i^t \mathbf{x})]\,x_j, \quad \text{for } j = 1, 2, \ldots, n$$

Note that this rule is applicable only for binary neuron response.
Under this rule, weights are adjusted if and only if o_i is incorrect.
Error as a necessary condition of learning is inherently included in this training rule. Obviously, since the desired response is either 1 or -1, the weight adjustment reduces to

$$\Delta \mathbf{w}_i = \pm 2c\mathbf{x}$$

where a plus sign is applicable when $d_i = 1$, and $\operatorname{sgn}(\mathbf{w}^t\mathbf{x}) = -1$, and a minus sign is applicable when $d_i = -1$, and $\operatorname{sgn}(\mathbf{w}^t\mathbf{x}) = 1$.
The weight adjustment is inherently zero when the desired and actual responses agree.

Example 2

This example illustrates the perceptron learning rule of the network shown in Figure below. The set of input training vectors is as follows:

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}, \quad \mathbf{x}_2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}, \quad \mathbf{x}_3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$

and the initial weight vector $\mathbf{w}^1$ is assumed identical as in Example 1. The learning constant is assumed to be $c = 0.1$. The teacher's desired responses for $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ are $d_1 = -1$, $d_2 = -1$, and $d_3 = 1$, respectively. The learning according to the perceptron learning rule progresses as follows.

Step 1 Input is $\mathbf{x}_1$, desired output is d_1:

$$net^1 = \mathbf{w}^{1t}\mathbf{x}_1 = \begin{bmatrix} 1 & -1 & 0 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = 2.5$$

Correction in this step is necessary since $d_1 \neq \text{sgn}(2.5)$. We thus obtain updated weight vector

$$\mathbf{w}^2 = \mathbf{w}^1 + 0.1(-1 - 1)\mathbf{x}_1$$

Plugging in numerical values we obtain

$$\mathbf{w}^2 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} - 0.2 \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix}$$

Step 2 Input is $\mathbf{x}_2$, desired output is d_2. For the present weight vector $\mathbf{w}^2$ we compute the activation value net^2 as follows:

$$net^2 = \mathbf{w}^{2t}\mathbf{x}_2 = \begin{bmatrix} 0 & 1.5 & -0.5 & -1 \end{bmatrix} \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix} = -1.6$$

Correction is not performed in this step since $d_2 = \text{sgn}(-1.6)$

Step 3 Input is $\mathbf{x}_3$, desired output is d_3, present weight vector is $\mathbf{w}^3$. Computing net^3 we obtain:

$$net^3 = \mathbf{w}^{3t}\mathbf{x}_3 = \begin{bmatrix} -1 & 1 & 0.5 & -1 \end{bmatrix} \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix} = -2.1$$

Correction is necessary in this step since $d_3 \neq \text{sgn}(-2.1)$. The updated weight values are

$$\mathbf{w}^4 = \mathbf{w}^3 + 0.1(1 + 1)\mathbf{x}_3$$

or

$$\mathbf{w}^4 = \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix} + 0.2 \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.6 \\ -0.4 \\ 0.1 \\ 0.5 \end{bmatrix}$$

This terminates the sequence of learning steps unless the training set is recycled. It is not a coincidence that the fourth component of $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$..

Summary of Learning Rules

Table 1 provides the summary of learning rules and of their properties.

Learning rule	Single weight adjustment Δw_{ij}	Initial weights	Learning	Neuron characteristics	Neuron / Layer
Hebbian	$co_i x_j$ $j = 1, 2, \ldots, n$	0	U	Any	Neuron
Perceptron	$c\left[d_i - \operatorname{sgn}(\mathbf{w}_i^t \mathbf{x})\right] x_j$ $j = 1, 2, \ldots, n$	Any	S	Binary bipolar, or Binary unipolar*	Neuron
Delta	$c(d_i - o_i)f'(net_i)x_j$ $j = 1, 2, \ldots, n$	Any	S	Continuous	Neuron
Widrow-Hoff	$c(d_i - \mathbf{w}_i^t \mathbf{x})x_j$ $j = 1, 2, \ldots, n$	Any	S	Any	Neuron
Correlation	$cd_i x_j$ $j = 1, 2, \ldots, n$	0	S	Any	Neuron
Winner-take-all	$\Delta w_{mj} = \alpha(x_j - w_{mj})$ m-winning neuron number $j = 1, 2, \ldots, n$	Random Normalized	U	Continuous	Layer of p neurons
Outstar	$\beta(d_i - w_{ij})$ $i = 1, 2, \ldots, p$	0	S	Continuous	Layer of p neurons

c, α, β are positive learning constants
S — supervised learning, U — unsupervised learning
* — Δw_{ij} not shown