SCIENCE LECTURE

An introduction to Artificial Neural Networks

Author: Guillermo Rios

.- Why Neural Networks?

Von Neumann type of computers can solve a lot of problems, but only if their solution can be formalized in terms of algorithms to be translated into machine instructions that CPUs can execute. However, many of the everyday tasks that humans take for granted, like recognizing simple patterns and generalizing those patterns of the past into future actions, are difficult to formalize algorithmically.

Neural Networks offer a radically different approach to problem solving: they seek patterns in the input data, even when no one knows they are there. This makes them ideal to automatize pattern recognition tasks.

Neural Networks are relatively crude electronic models based on the structure of the brain. They involve the creation of massive parallel networks and the *"training"* of those networks to solve specific problems. Neural Networks are not *"programmed,"* they *"learn."*

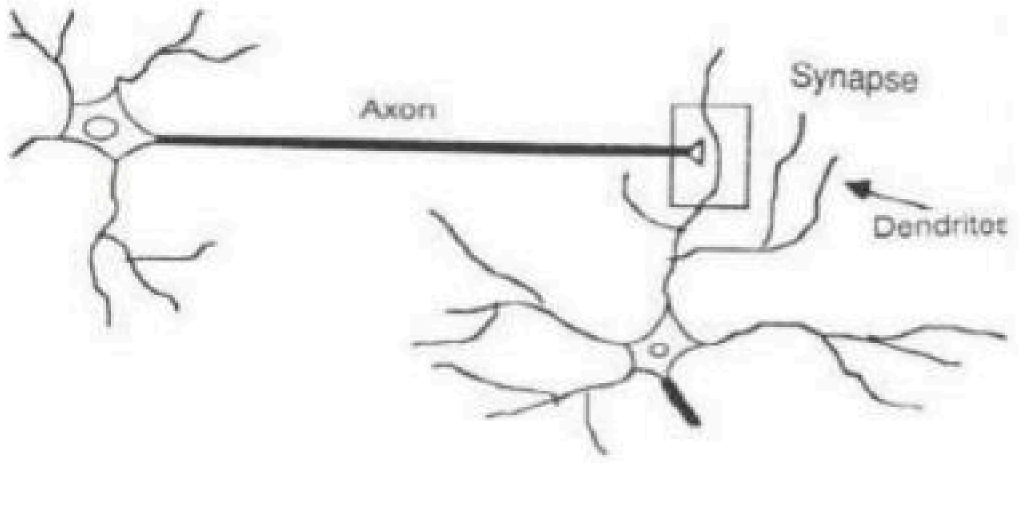
The working of the brain is still a mystery, but some of its aspects are known. The most basic element of the brain is a specific type of cells called neurons. It is assumed that neurons provide us with our abilities to remember, think and learn. Each one of these 100 billion cells is typically connected to between 1,000 to 10,000 other neurons.

The term "*neural network*" is in fact biological. What we call "*neural networks*" in Computer Science are actually "*artificial neural networks*" or ANN, which attempt to mimic these biological structures both in architecture and operation, at least to the extent that we know them.

The idea of ANN has been around since the 1940s, but only in the late 1980s were they advanced enough to prove useful.

.- The biological neuron.

There are many types of neurons, but they all share basic common elements, some of which are shown in Fig. 1.



Important parts of a typical neuron:

<u>Synapses:</u> The electrochemical contact between neurons.

<u>Dendrites:</u> Accept inputs from another neurons.

<u>Soma:</u> Processes the inputs.

<u>Axon:</u> Takes the neuron output to another neurons.

<u>Axon hillock:</u> Makes sure that only those signal that are strong enough can pass.

Fig. 1

Synapses are connections between neurons. They are not physical connections, but miniscule gaps that allow electric signals to jump across from neuron to neuron. These electrical signals are then passed to the soma, which performs some operation and sends out its own electrical signal to the axon, through the Axon hillock. The Axon hillock acts like a filter ensuring that only those signals that are strong enough are passed to the Axon. The axon then distributes this signal to Dendrites. Dendrites carry the signals out through the various synapses to another neurons, and the cycle repeats, like in a chain reaction.

In the brain, learning occurs by changing the effectiveness of the synapses so the influence of one neuron on another changes.

.- A basic artificial neuron.

Just as there is a basic biological neuron, there is basic artificial neuron, which is an electronic cell that mimics the functionality of a biological neuron.

This functionality is captured in the artificial neuron below known as the Threshold Logic Unit (Fig 2,) like the originally proposed by McCulloch and Pitts in 1943.

Artificial Neuron

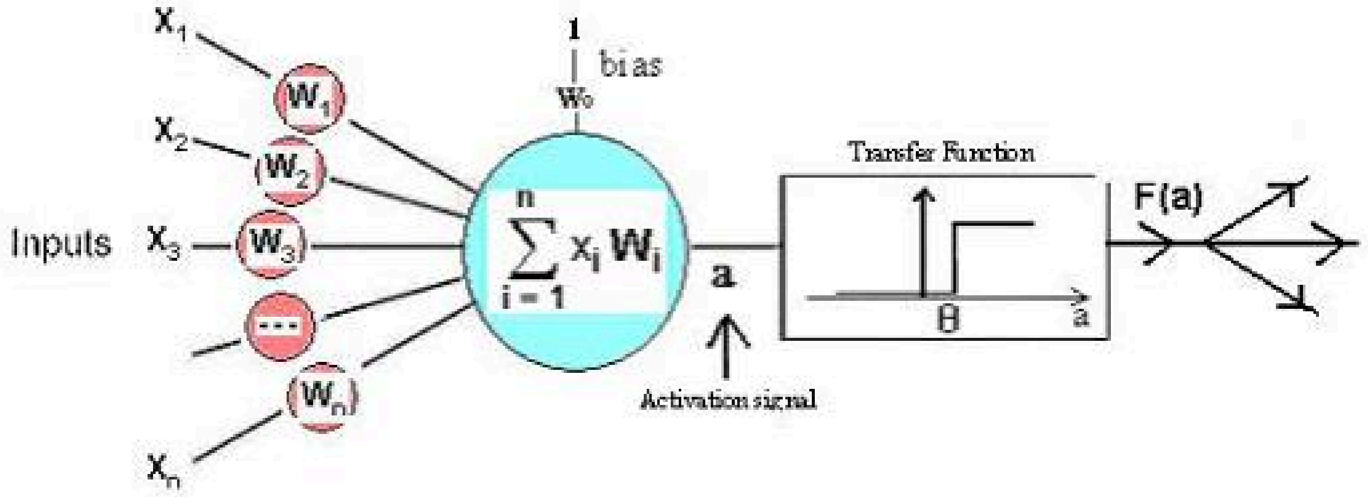


Fig. 2.

The meaning of this intimidating diagram is actually quite simple.

The artificial soma (in blue) receives several input signals x_k , for k = 1, 2… n, which may come from another artificial neurons or perhaps an electrical sensor device.

Before entering, each signal is multiplied by its connection weight W_k (in red.) The purpose of this weight is to make some of the input signals more important than others, as it happens with real neurons. Thus, the effectiveness of the brain synapses is modeled by a modifiable weight value $W_{k,}$ which is associated with each connection. Later we will see that ANNs "learn" by changing the values W_k for all neurons in the network.

The immediate result of this input, the so-called activation signal "a", is computed as the weighted sum $a = x_1 W_1 + x_2 W_2 + \dots X_n W_n$.

The activation signal is then transformed by the Transfer Function, which plays the role that the Axon hillock plays in real neurons.

In Fig 2 above, the Transfer Function F(a) shown is a "Step Function," and its effect is quite simple: if "a" is larger or equal to a given threshold value θ, the final output F(a) is an electrical voltage equivalent to 1, otherwise F(a) is zero.

The weights in most neural networks can be both negative and positive, therefore providing excitatory or inhibitory influences to each input. Also, sometimes a bias value W_0 is included, making $a = x_1 W_1 + x_2 W_2 + \dots X_n W_n + W_0$. This W_0 provides a convenient way to account for the threshold, by making $x_0 W_0 = - \theta$.

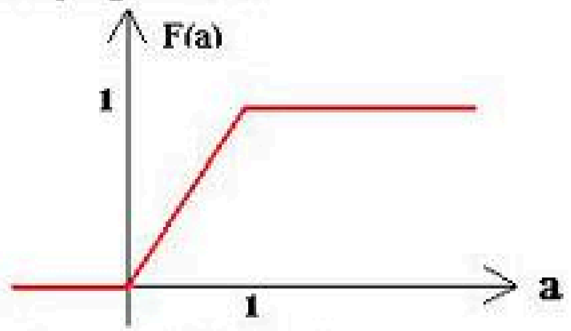
.- More general artificial neurons.

The electronic implementation above is actually just one of many possibilities. Artificial neurons can be constructed utilizing different summing functions as well as different transfer functions.

Alternatives to the weighted sum are the average, the largest or the smallest input, Boolean operations AND, OR, etc. on the inputs, etc.

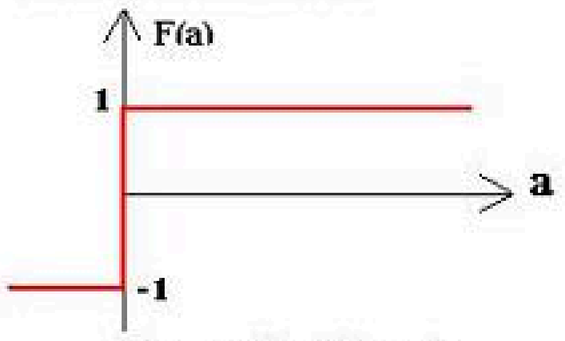
As for the Transfer Function F(a), typical alternatives are:

Ramping function



For $a < 0$, $F(a) = 0$
For $0 <= a <= 1$, $F(a) = a$
For $a > 1$, $F(a) > 1$

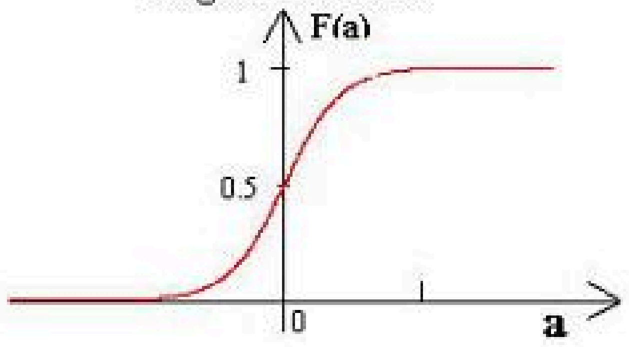
Hard Limiter function



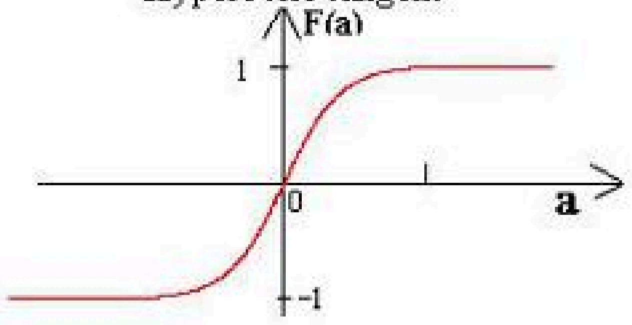
For $a < 0$, $F(a) = -1$
For $a >= 0$, $F(a) = +1$

and the following "S" shaped functions:

Logistic Function



Hyperbolic tangent



$$F(a) := \frac{1}{1 + e^{-\rho \cdot a}}$$

$$F(a) := \frac{e^{\rho \cdot a} - e^{-\rho \cdot a}}{e^{\rho \cdot a} + e^{-\rho \cdot a}}$$

where ρ is the "Gain" parameter, which determines how abrupt the transition is from the region $a<0$ to the region $a> 0$.

The term "Perceptron" is often used to refer to artificial neurons. However, it is a term widely used in neural networking and there seems not to be a single definition for it.

.- A Network of Artificial Neurons.

Even though artificial neurons can be combined in a myriad of ways, a neural network is more than just a bunch of connected neurons. Even the brains of very simple animals are structured devices, and so must be the artificial neural networks. ANNs are typically organized in layers.

The functioning of an ANN is determined by four elements: a) The organization of the neurons into layers b) The summation function c) The transfer function d) The weights values given to the connections between the layers.

In an ANN we can identify the input layer, the output layer, and one or several layers in between, known as the hidden layers, as in Fig. 3.

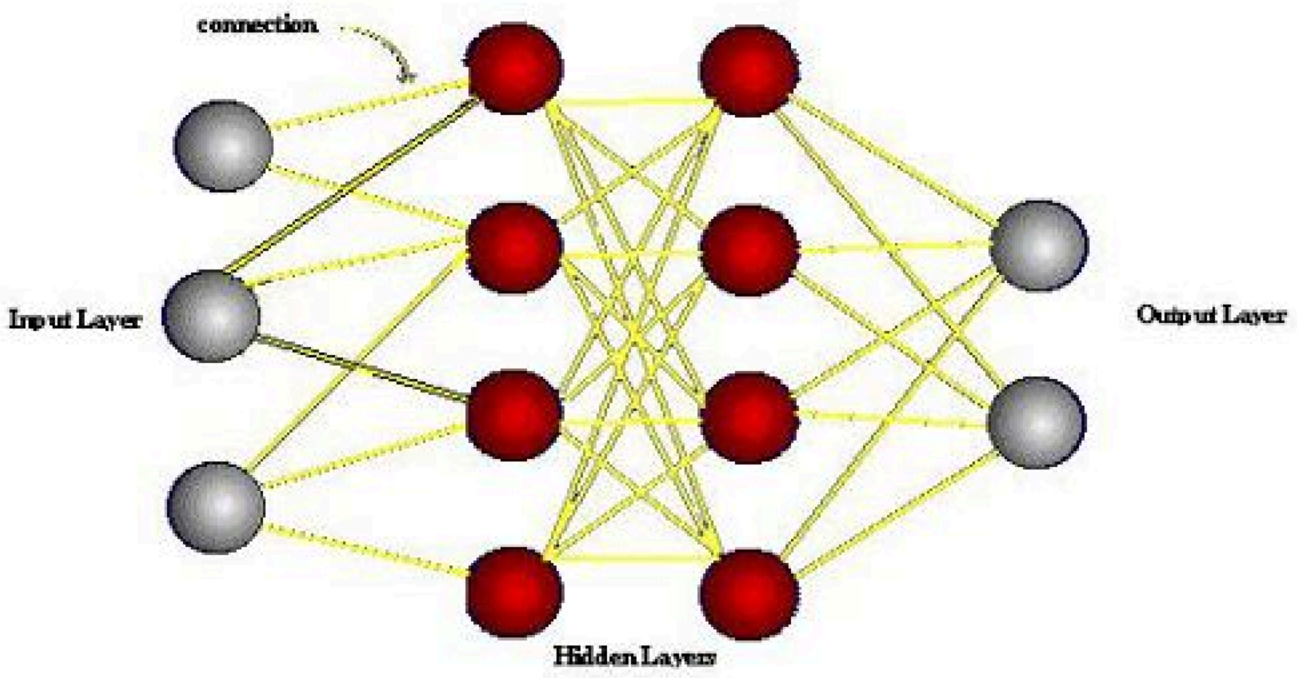


Fig. 3.

The parallel processing that typically takes place in an ANN is as follows.

Patterns are presented to the network via the "input layer" which communicates to the hidden layers, where the actual processing is done via the system of weighted connections. The hidden layers eventually link to an output layer where the answer is output.

The input layer receives its data either from input files or directly from electronic sensors in real-time applications. The output layer sends information directly to the outside world, to a secondary computer process, or to other devices such as mechanical control systems.

Once the structure, the summing and the transfer functions have been selected, for a given ANN to provide the "correct" answer at recognizing each possible input, the weight values of all the connections must be adjusted. In other words, the ANN must "learn" or be "trained."

As the reader can imagine, the assigning of such values by hand would be a humongous task for ANN larger than just a few neurons. So, an automatic procedure to find the right W_k's must be provided. In other words, a "learning" algorithm must be established.

.- The "training" of a neural network.

The learning algorithm is the rule by which all the weights across the ANN are modified in the training stage, according to the input patterns that it is presented with.

In a sense, ANNs learn by example, as do their biological counterparts. There are many different kinds of learning algorithms used on neural networks. Following we describe one frequently used.

A pattern that we want to teach the ANN to recognize is presented to the input layer. These values are then propagated towards the output layer as described earlier.

The result obtained at the network's output is most probably erroneous, especially at the beginning of the training, when the initial values of the weights were probably randomly assigned (although in a suitable range, e.g. –1 to 1.) The "error value" is then computed as the difference between the expected, "right" value, and the actual output value. This error value is then "back-propagated" by going in reverse towards the input layer while modifying the weights proportionally to each one's contribution to the total error value.

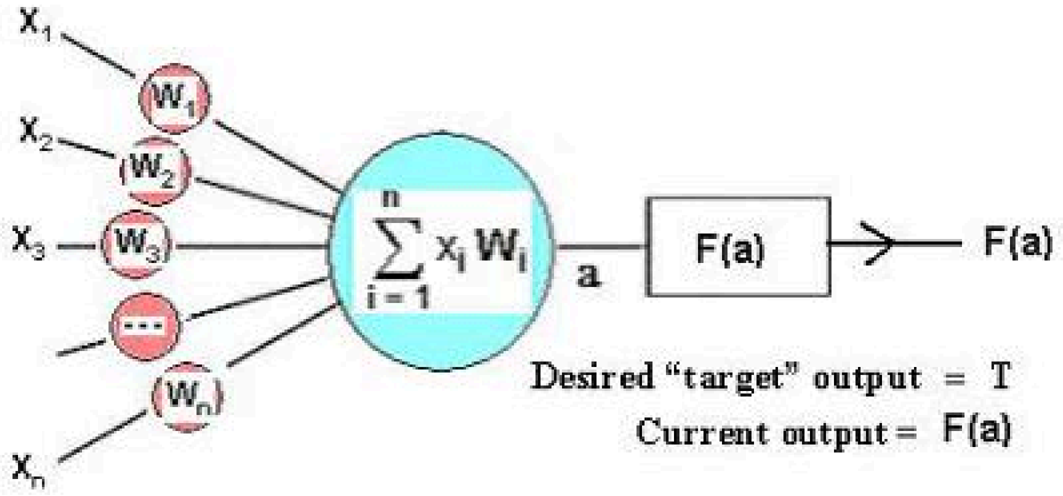
This procedure is repeated for the pattern until the overall error for this particular pattern is acceptably small.

The same procedure is repeated for each pattern we want to teach the ANN to recognize. If the number of neurons in the ANN is enough for the number of different patterns, this iterative learning process (this is, the adjustment of all the W_k's) will converge.

.- A simple example.

This section will require differential calculus. It may be skipped if that is too big of a challenge.

Let's apply the back-propagation algorithm for the simplest example: a single neuron like the one below.



From the given inputs x_k and the present values of the weights W_k, our only neuron (we mean the example's, not our own) produces an output F(a). Let's assume that this output is substantially different from the desired target output **T**, so we want to "teach" our neuron by adjusting the W_k's to reduce this error.

A positive error is as bad as a negative one, so we do not want to just minimize the difference $\mathbf{T} - F(a)$ but its absolute value $|\mathbf{T} - F(a)|$.

Since minimizing $|\mathbf{T} - F(a)|$ is equivalent to minimizing $(\mathbf{T} - F(a))^2$, let's define the more manageable error function:

$$E(a) = ½\,(\mathbf{T} - F(a))^2$$

were the factor ½ has been included for algebraic convenience, and where $a = x_1W_1 + x_2W_2 + \ldots X_nW_n$. The x_k's are constant since the input is fixed. What we want is to vary the W_k's to make the output F(a) close enough to the desired target **T**. Thus, we have:

$$E(a) = E(W_1, W_2, \ldots W_n) = ½\,(\mathbf{T} - F(W_1, W_2, \ldots W_n))^2 \qquad \textbf{Eqt. 1}$$

In other words, E is a function in the n-dimensional vector space $W_1 \times W_2 \times \ldots W_n$. We can take the differential of this function $E(W_1, W_2, \ldots W_n)$ as

$$dE = \frac{\partial E}{\partial w_0} dw_0 + \frac{\partial E}{\partial w_1} dw_1 + \cdots \frac{\partial E}{\partial w_n} dw_n$$

were we used the standard symbol ∂ to denote the partial derivative respect one of the variables. This differential can be expressed as the scalar product of two vectors in the R^n space $W_1 \times W_2 \times \ldots W_n$:

$$dE = \left[\frac{\partial E}{\partial w_0}, \ \frac{\partial E}{\partial w_1}, \ \cdots \ \frac{\partial E}{\partial w_n} \right] \cdot \left[dw_0, \ dw_1, \ \cdots \ dw_n \right]$$

We can write this equation in a more convenient way by using the so-called gradient vector notation:

$$\nabla E(w_0, w_1, \cdots w_n) \equiv \left[\frac{\partial E}{\partial w_0}, \ \frac{\partial E}{\partial w_1}, \ \cdots \ \frac{\partial E}{\partial w_n} \right]$$

Thus,

$$dE = \nabla E(w_0, w_1, \cdots w_n) \cdot \left[dw_0, \ dw_1, \ \cdots \ dw_n \right]$$

Now, for an infinitesimal change ($dW_1, dW_2, \ldots dW_n$) the maximum rate of change of E occurs when the vector ($dW_1, dW_2, \ldots dW_n$) points towards the direction of vector ∇E. If we want to reduce E then we need to change the W_k's in the direction opposite to ∇E.

We cannot make infinitesimal changes on the W_k's, but finite ones. Let's call them ΔW_k. Thus, we want

$$\Delta \vec{W} = -\eta \ \nabla E \left(w_0, w_1, \cdots w_n \right) \quad \text{or} \quad \Delta W_k = -\eta \frac{\partial E}{\partial w_k}$$

where η is a small number to be selected based on how much we should move in order to obtain a convenient change in the W_k's.

We see from Eqt. 1 that to derive the function $\partial E / \partial w_k$ we need to know the neuron's transfer function F(a). For this example we will assume the "S" shaped logistic function, which is frequently used:

$$F(a) = \frac{1}{(1 + e^{-a})} \quad \text{or} \quad F(a) = \frac{1}{1 + e^{-(x_1 W_1 + x_2 W_2 + \ldots X_n W_n)}}$$

This function bears a greater resemblance to real neurons. Thus, for $\partial E / \partial w_k$ we have

$$\frac{\partial E}{\partial w_k} = \frac{\partial}{\partial w_k}\left(\frac{1}{2}(\ T - F(a)\ )^2\right)$$

$$\frac{\partial E}{\partial w_k} = \frac{1}{2}\frac{\partial}{\partial w_k}(\ T - F(a)\ )^2$$

$$\frac{\partial E}{\partial w_k} = \frac{1}{2}\,2(\ T - F(a))\ \frac{\partial}{\partial w_k}(\ T - F(a)\ )$$

Since T is the fixed target value, it is a constant, so we have

$$\frac{\partial E}{\partial w_k} = -(T - F(a))\ \frac{\partial}{\partial w_k} F(a)$$

where the partial derivative of F(a) can be computed as follows:

$$\frac{\partial}{\partial w_k} F(a) = \frac{dF(a)}{d\,a}\frac{\partial\,a}{\partial w_k} = \frac{d\,F(a)}{d\,a}\,x_k$$

For our particular $F(a) = 1/(1 + e^{-a})$ the user can verify that

$$\frac{dF(a)}{d\,a} = -\frac{1}{(1 + e^{-a})^2}(-e^{-a}) = F(a)\cdot(1 - F(a)\ )$$

Using these results we have

$$\frac{\partial E}{\partial w_k} = -\,x_k(T - F(a))\cdot F(a)\cdot(1 - F(a))$$

and finally

$$\Delta W_k = -\eta\frac{\partial E}{\partial w_k} = \eta\ x_k(\ T - F(a)\ )\cdot F(a)\cdot(1 - F(a)) \qquad \textbf{Eqt. } 2$$

Eqt 2 specifies the change to be made to each W_k to get their new values. With these new W_k's we test if the error function is less than a maximum error E_{max} that we are willing to tolerate:

$$E(a) = \frac{1}{2} \left(\mathbf{T} - F(W_1, W_2, \dots W_n) \right)^2 < E_{max}$$

If it is not, then another iteration of the process must be executed.

Eqt. 2 shows the importance of selecting a convenient value for η. With a learning rate η that is too large, the updated W_k's overshoot the minimum error E leading to oscillations about the minimum. On the other hand, a learning rate η that is too small causes slow convergence.

.- Unsupervised training.

A learning algorithm like the one above, in which a target output is provided, is known as "supervised" training.

There is another type of training known as Unsupervised Learning, in which the network is able to discover statistical regularities in its input and automatically develop different modes of behavior to represent different classes of inputs. They are trained by letting the network continually adjust itself to new inputs by finding relationships within the data as this is presented and automatically defining classification schemes.

Thus, for an unsupervised learning rule, the training set consists of input training patterns only. Therefore, the network is trained without benefit of any teacher. The network learns to adapt based on the experiences collected through the previous training patterns.

In general, to make determinations neural networks may use many other principles, such as fuzzy[1] logic and genetic algorithms[2].

.- Conclusion.

If we have succeeded, this introduction to Artificial Neural Networks has presented the topic in a deceptively simple way. Even though fascinating, the field of ANN is ample and complex.

[1] Fuzzy logic is an approach to computing based on "degrees of truth" rather than the usual "true or false" (1 or 0) Boolean logic.

[2] Genetic Algorithms (GA) are methods of "breeding" computer programs and solutions to optimization or search problems by means of simulated evolution. Processes loosely based on natural selection, crossover, and mutation are repeatedly applied to a population of binary strings which represent potential solutions. Over time, the number of above-average individuals increases, and better fit individuals are created, until a good solution to the problem at hand is found.

Neural Networks is not only an area of vanguard scientific research, but also of many thriving industrial and commercial applications. It is not surprising that ANN becomes as ubiquitous as von-Neumann-type computers are.

Some commercial and industrial applications include Quality Control, Financial Forecasting, Economic Forecasting, Data Mining (discovering of patterns in data bases,) Credit Rating, Speech & Pattern Recognition, Biomedical Instrumentation, Process Modeling & Management, Laboratory Research, Oil & Gas Exploration, Reduction, Targeted Marketing, Bankruptcy Prediction, Machine Diagnostics, Securities Trading, Signature Analysis, Process Control, etc.

There is plenty material in the Internet on neural networks. A few of our diagrams were downloaded and edited. The addresses of those web pages are included in the references below to give them due credit.

Some of these references offer shareware programs for PCs and Macs that simulate ANNs at a different degree of complexity.

.- References.

Disclaimer: This is a list of references that were relevant at the time (year 2002.) Now, many years later, many of these links may not even exist.

ftp://ftp.sas.com/pub/neural/FAQ.html
http://www.generation5.org
http://www-gpi.physik.uni-karlsruhe.de/pub/robert/Diplom/Diplom.html
http://vv.carleton.ca/~neil/neural/neuron.html
http://neil.fraser.name/software/
http://www.inference.phy.cam.ac.uk/mackay/itprnn/ps/480.490.pdf
http://www.cse.unsw.edu.au/~billw/cs9414/notes/ml/backprop/backprop.html
http://www.pmsi.fr/neurin2a.htm
http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html
http://www.shef.ac.uk/psychology/gurney/notes/l1/section3_1.html
http://home.ipoline.com/~timlin/neural/NeuralNetwork.htm
http://www.gc.ssr.upm.es/inves/neural/ann1/anntutorial.html
http://www.cs.bgu.ac.il/~omri/NNUGA/
http://www.cs.swarthmore.edu/~meeden/nnet/bp-derivation/backprop.html
http://q.cis.uoguelph.ca/~skremer/Teaching/27476/lectures/lectures.html
http://www.indiana.edu/~gasser/Q351/bp_derivation.html
http://www.indiana.edu/~gasser/Q351/ff2.html
http://www.maths.uwa.edu.au/~rkealley/ann_all/
http://richardbowles.tripod.com/neural/glossary/glossary.htm
http://www.emsl.pnl.gov:2080/proj/neuron/neural/
http://blizzard.gis.uiuc.edu/htmldocs/Neural/neural.html