Neural networks

Introduction

The process of information in a computer is done in different way than in human brain, this lead to facts that computer and brain have their own strengths and weakness in application, hence we will attempt to build computers which copy the activities of the brain.

In order to understand all the concepts of neural networks, it is necessary to concern with the working of human brain, this will be done in very simple way because only a basic understanding is required.

In the beginning

The human neuron system consists of networks of highly interconnection neurons, each of which performs a computation at any given moment. The results are transmitted to other neurons a long of pathway. A single neuron can send the result to as many as 10000 other neurons as a signal to the inputs of other neuron s in the form of voltages. The voltage can either inhibit the other neurons from sending signals or excite them to sending to other neurons along the pathway.

The neurons are the actual processing units in the brain like every processor they possess an input and an output. The input signals are electrochemical stimulus which comes from other neurons and are transmitted to the neuron through special lines. The dendrites are input lines of neuron and each one has 10000 of dendrites. The neuron body may process specific dendrites differently than others, for example it may add or subtract a constant voltage (called Bias) from a dendrite. There is one Axon for each neuron which is carries the fired voltage from neuron body to many connections with other neurons, these connections are called Synapses, and there are thousands of synapses connected to a single axon the synapse connection determine the amount of signal reach to the dendrite this value called “weighting factor” as in fig 1.

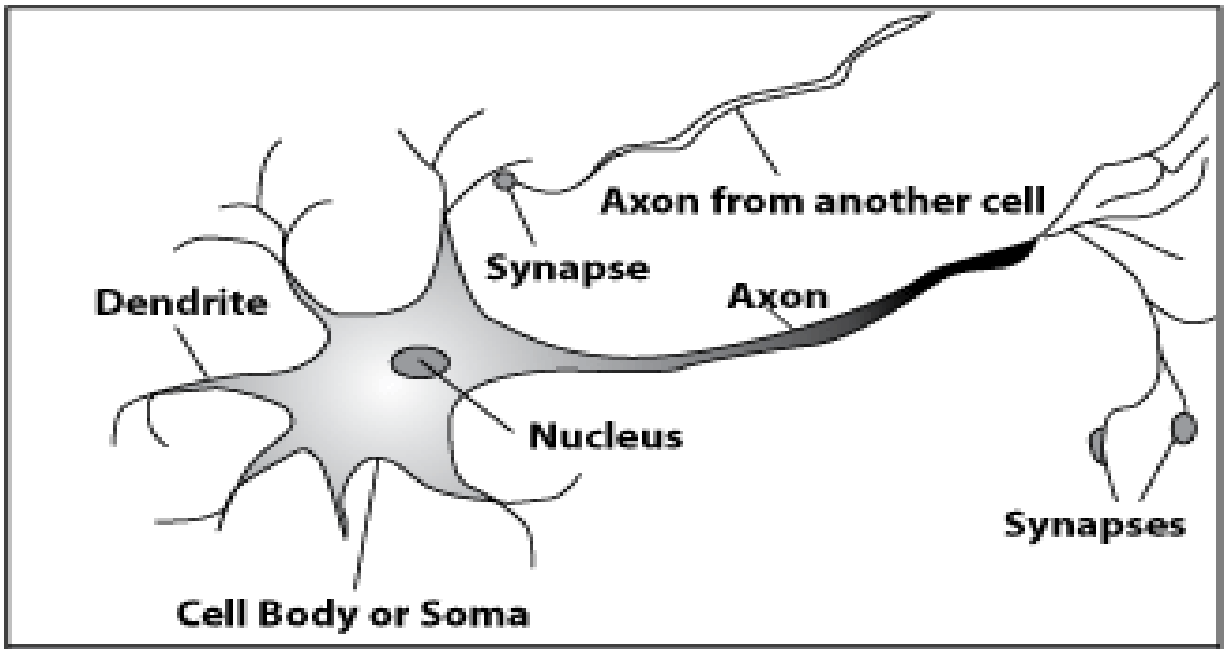


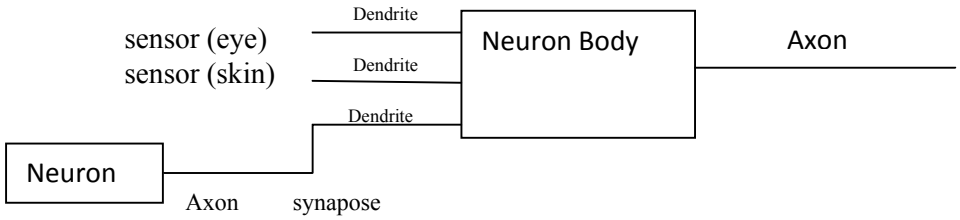
Figure 1: A Biological Neuron

Training the system

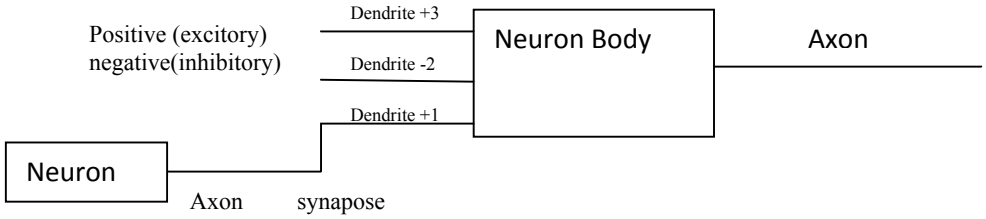
When the human neural system sees an object, some of the eye sensors are activited. This sensors fire, sending signals to hidden neurons, neurons that fired during the same training session increase the connection strength between them. The same time the objects appears, the approprate connections have been strength. When another object appears, the neural system would have to be trained to recognize that object.

Modeling the single neuron

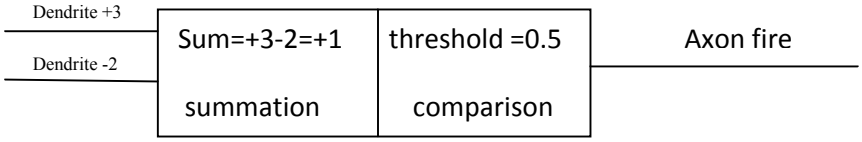
The neuron is an electronic device that responds to electrical signals, the single neuron model consists of dendrites, the neuron body, an axon, and synaposes as follow:



The dendrite signals can have positive or negative voltages, the positive voltage contribute to exciting the neuron body to sending a signal while the nagitive voltage will inhibiting the neuron body from sending a signal show the following:



The neuron body is the computer that process a signals carried by dendrites, the neuron body model that we will consider sums the signals carried by its dendrites, if the sum exceed the threshold level, the neuron body will fire, otherwise the neuron body is not fire as follow:



Note: the neuron body may process specific dendrites differently than the others, it may add or subtract a constant voltage from dendrites.

An Artificial Neuron

The artificial neuron simulates four basic functions of a biological neuron. Fig 3 shows basic representation of an artificial neuron.

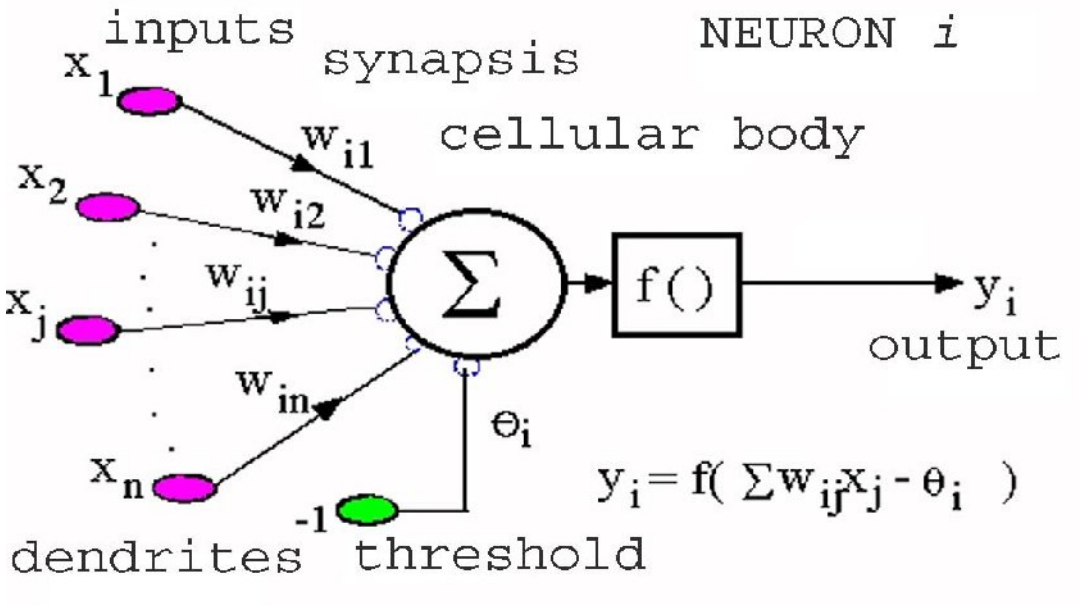


Fig. 3: A basic artificial neuron

In Fig. 3, various inputs to the network are represented by the mathematical symbol, x(n). Each of these inputs is multiplied by a connection weight. The weights are represented by w(n). In the simplest case, these products are summed, fed to a transfer function (activation function) to generate a result, and this result is sent as output. This is also possible with other network structures, which utilize different summing functions as well as different transfer functions. Some applications like recognition of text, identification of speech, image recognition etc. Seven major components make up an artificial neuron. These components are valid whether the neuron is used for input, output, or is in the hidden layers.

Component 1. Weighting Factors: A neuron usually receives many simultaneous inputs. Each input has its own relative weight, which gives the input the impact that it needs on the processing element's summation function. Some inputs are made more important than others to have a greater effect on the processing element as they combine to produce a neural response.

Component 2. Summation Function: The inputs and corresponding weights are vectors which can be represented as (i1, i2 . . . in) and (w1, w2 . . . wn). The total input signal is the dot product of these two vectors. The result; (i1 * w1) + (i2 * w2) +…….. + (in * wn) ; is a single number. The summation function can be more complex than just weight sum of products. The input and weighting coefficients can be combined in many different ways before passing on to the transfer function. In addition to summing, the summation function can select the minimum, maximum, majority, product or several normalizing algorithms.

Component 3. Transfer Function: The result of the summation function is transformed to a working output through an algorithmic process known as the transfer function. In the transfer function the summation can be compared with some threshold to determine the neural output. If the sum is greater than the threshold value, the processing element generates a signal and if it is less than the threshold, no signal (or some inhibitory signal) is generated. It is called a sigmoid when it ranges between 0 and 1, and a hyperbolic tangent when it ranges between -1 and 1.

Component 4. Scaling and Limiting: After the transfer function, the result can pass through additional processes, which scale and limit. This scaling simply multiplies a scale factor times the transfer value and then adds an offset. Limiting is the mechanism which insures that the scaled result does not exceed an upper, or lower bound. This limiting is in addition to the hard limits that the original transfer function may have performed.

Component 5. Output Function (Competition): Each processing element is allowed one output signal, which it may give to hundreds of other neurons. Normally, the output is directly equivalent to the transfer function's result. Some network topologies modify the transfer result to include competition among neighboring processing elements.

Component 6. Error Function and Back-Propagated Value: In most learning networks the difference between the current output and the desired output is

calculated as an error which is then transformed by the error function to match a particular network architecture.

Component 7. Learning Function: Its purpose is to modify the weights on the inputs of each processing element according to some neural based algorithm.

For example

The problem is to construct a neural network (Pitts neural network) to associated human physical characteristic such as height and weight. This may not seem to be a very complicated problem but we will be able to use it to design and solve problem in other domain such as "image recognition". The following table identify four persons together with their characteristic:

Input	person	Person1	Person2	Person3	Person4
	output	01	10	11	00
Mustache (yes=1,no=0)		1	0	1	1
Glasses (yes=1,no=0)		1	0	0	1
Hair color (dark=1,light=0)		0	0	1	1
Weight(heavy=1,light=0)		1	1	0	0
Height(tall=1,small=0)		0	1	1	0

Table 1: four persons with their characteristic

Design of neural network

The neural networks solve this problem will contain five inputs, one for each characteristic. We can referred for each person by two binary digits, two neurons are needed to provide the output one for each digit. The configuration of the neural network will contain one level of neurons which connects to both the input and output as show in figure 4.

Training the network

Next we must train the network to the specific characteristic inputs to person associated with those inputs, the characteristic inputs of person 1 as in table 1

11010 (mustache, glasses, light hair, heavy, small)

When receiving those inputs the network “ideal” output would be
01 (identify as person 1)

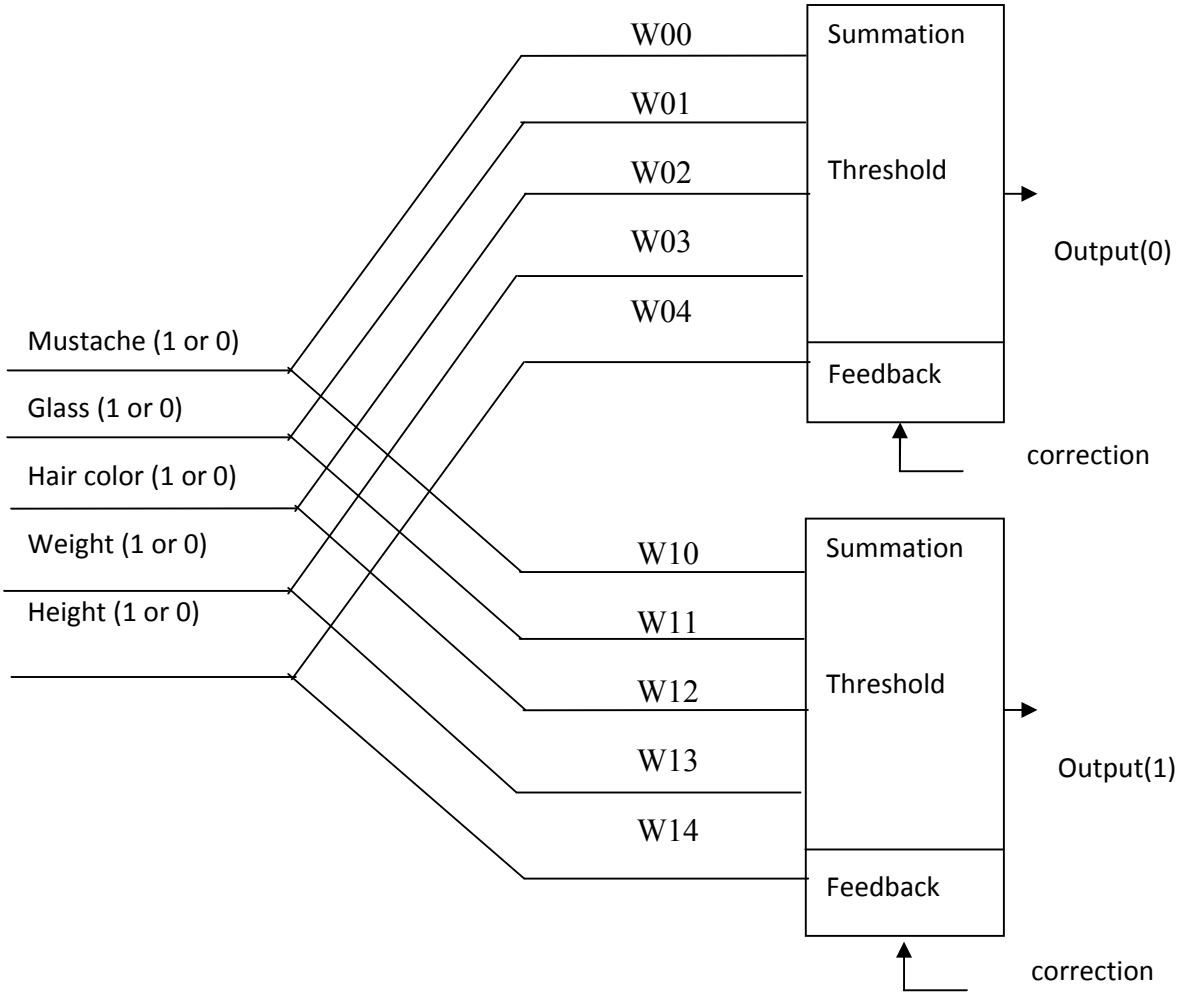


Fig. 4: a neural network to identify the 4 persons

If the “actual” output is not 01, the network trainer will responds with
output error=1
at each neuron which is in error, for example if the network should have identified person 1 but instead produced an output of:
11 (identified as person 3)
Neuron (1) is in error since its output should have been 0

This method of training is the way of people learn, if a response is an error, the person attempts to modify the reasoning process so as to eliminate the error. If the response is correct, no think happens, similarly when an output error is presented by a neuron, it will begin to self-modification by modifying its weight values to correct the error.
Let us started with a step by step explanation the training method used to modify the weight values.

Forward propagation

The first phase in the training session is to apply an input and calculate the resulting output, this is called "forward propagation" it is the process of flowing from input to output.

Apply the input

We will input the characteristics of person1 as follow before that we will input the weight values for each neuron=1 as follow:
Enter weighting value for neuron 0 input 0 ?=1
Enter weighting value for neuron 0 input 1 ?=1
.
.
.
Enter weighting value for neuron 1 input 4 ?=1

The characteristic of person1

Does person has mustache Yes=1;No=0 ? 1
Does person has glasses Yes=1;No=0 ? 1
Does person has dark heir Yes=1;No=0 ? 0
Is person heavy Yes=1; No=0 ? 1
Is person tall Yes=1;No=0 ? 0

Calculating the sum for each neuron

Each input signal is multiplied by the weighting value associated with the input as shown below :

Input[0]*weight[0,0]
Input[1]*weight[0,1]
The neuron's sum unit will then add all the products associated with each input to arrive at a sum. Each neuron in the network will calculate its own sum as:
Sum[0]=3
Sum[1]=3

Comparison with threshold
Each neuron has threshold. If the sum exceed threshold, the neuron is fired represent (1) output else the neuron not fired represent (0) output as:

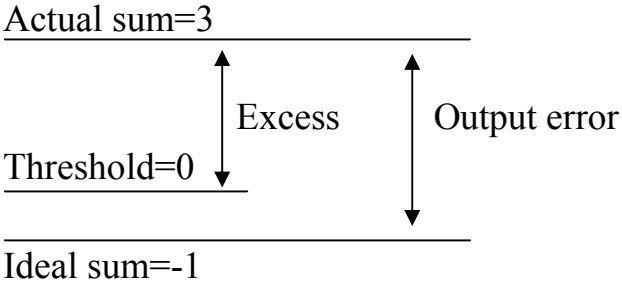
If sum >threshold then fire output=1
If sum<= threshold then not fire output=0

Backward Propagation
The output of each neuron is referred to as the "actual output". The "ideal output" is the output neuron expected to have based upon the current input, if the actual and ideal outputs are not equal an "error" has been detected, the network must "go backward" from the output toward the input correcting the weighting values to reduce or eliminate the error.

Is the output correct
The first step of back propagation is to determine if there is an output error by comparing every neuron's actual output with the ideal output. For the above example, the person1 has ideal output output[0]=1, output[1]=0 (from the table) and the actual output is output[0]=1, output[1]=1. The trainer responded with an error for output [1] and no error for the output[0]. The network has now been informed that all weight values contributing to neuron[1]'s output must be corrected to reduce or eliminate the error.

Calculating the error
The second step is to determine the size of the error. An output error will result in one of two ways, one way is when the sum exceeds the threshold causing the neuron to fire when it should not have fired as its happened in our example when ideal output is 0 while actual output is 1 for output[1]. The other way is reverse.

Let us consider the case where the sum exceeded the threshold and is an error as:

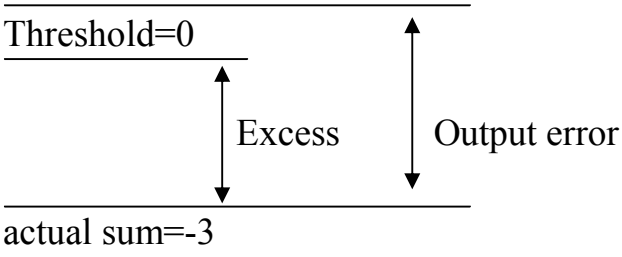


The actual sum exceeds the threshold by 3, the ideal sum is below the threshold would inhibit firing, the output error is define:

Output error=actual sum-ideal sum

For our example: output error[1]=3-(-1)=4

The alternative case occurs when the sum does not exceed the threshold when ideally it should have as shown below:



the actual sum is less than the threshold, the output error for this section
-3-(+1)=-4

Changing the weight values

If the sum of neuron is in error, we must determine the degree of responsibility that each input had in contributing to the error. The name we give this degree of responsibility is "Blame" we attach a blame value to each input that may have contributed to the error. The weights for those inputs with the largest blame values will receive the largest change.

Another factor affecting the degree of weight change is the size of output error. The greater the neuron's output error, the greater each of the neuron's weight must change. Therefore, the change in neuron's input weight values is proportional to

both the neuron's output error and the input's blame value. The following equation will be used to calculate a "new weight value" at the neurons input:

Weight.new[n,i]=weight.old[n,i]-(B*blame[n,i]*output.error[n])

Where:
n is neuron number
i is the input number
B is the learning factor, B is one of the variables that control the rate at which the final corrected values are reached. The value of B in our example is equal to 0.5. you can experiment with different values of B which modify the rate at which the neuron reaches its true values.

Calculating the Blame
The Blame is the percentage contribution that an input to a neuron has in forming the output error. The contribution of an input to a neuron's output error is:

Contribution[n,i]=input value[n,i]*weight[n,i]
The Blame is:
Blame=Contribution[n,i]/sum[n]

Is the training over ?
After many training cycle you are ready to tryout the network. During tryout there is no error correction, to do this, select a sequence of inputs and record the number of output errors. If the network responds correctly a percentage of the time that meets your criteria. You have finished. If not, you must examine your training sequences or the design of the neural network to correct any differences.

Neural Network types can be classified based on following attributes:

I-Applications
1-Classification
2-Clustering
3-Function approximation
4-Prediction

II-Topology

1-Single layer

2-Multilayer

3-Recurrent

III-Learning Methods

1-Supervised

2-Unsupervised

Types of neural networks based topology :

1-Single layer feed forward network

A neural network in which the input layer of source nodes projects into an output layer of neurons but not vice-versa is known as single feed-forward network. In single layer network, 'single layer' refers to the output layer of computation nodes as shown in Fig. 5:

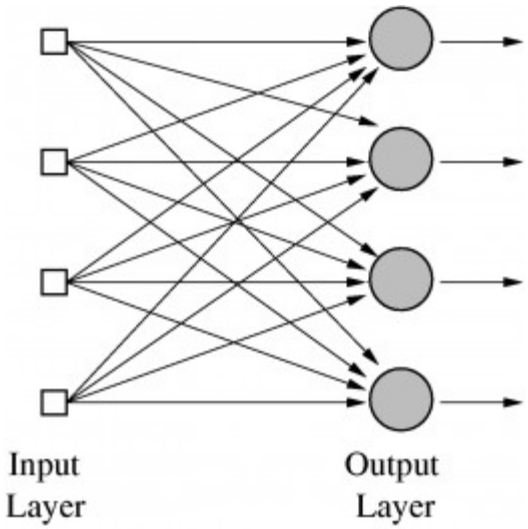


Fig. 5: single layer feed forward network

2-Muiltilayer feed forward network

This type of network consists of one or more hidden layers, whose computation nodes are called hidden neurons or hidden units. The function of hidden neurons is to interact between the external input and network output in some useful manner and to extract higher order statistics. The source nodes in input layer of network

supply the input signal to neurons in the second layer (1st hidden layer). The output signals of 2nd layer are used as inputs to the third layer and so on as in fig. 6.

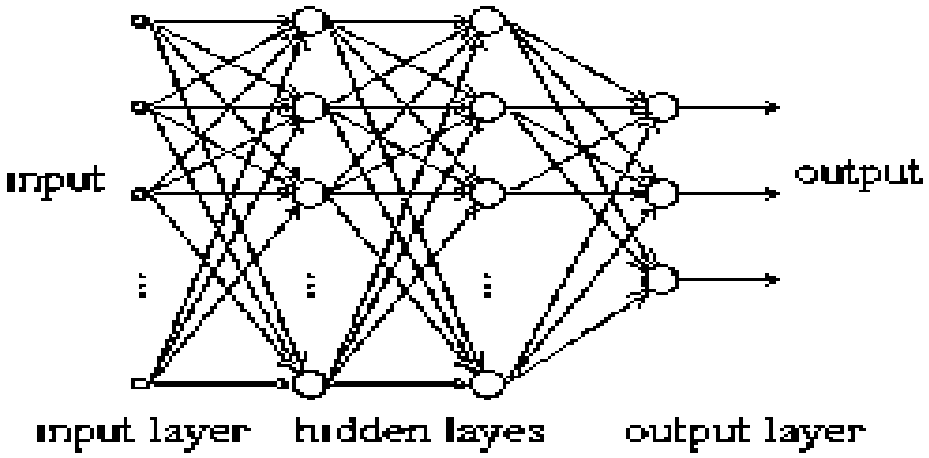


Fig. 6:A multilayer feed forward network

the overall response of network to the activation pattern supplied by source nodes in the input first layer. Short characterization of feed forward networks:

1. typically, activation is fed forward from input to output through 'hidden layers'.
2. mathematically, they implement static input-output mappings.
3. most popular supervised training algorithm: back-propagation algorithm
4. have proven useful in many practical applications as approximates of nonlinear functions and as pattern classification.

3-Recurrent network

A feed forward neural network having one or more hidden layers with at least one feedback loop is known as recurrent network as shown in Fig. 7 The feedback may be a self feedback, i.e., where output of neuron is fed back to its own input. Sometimes, feedback loops involve the use of unit delay elements, which results in nonlinear dynamic behavior, assuming that neural network contains non linear units.

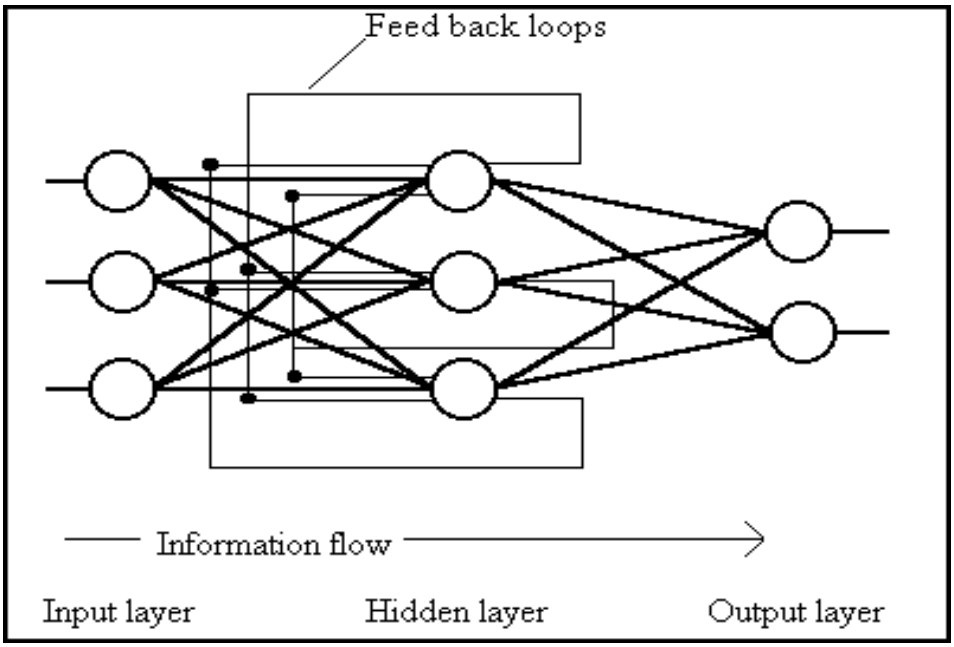


Fig. 7 : A recurrent network

There are various other types of networks like; delta-bar-delta, Hopfield, vector quantization, counter propagation, probabilistic, Hamming, Boltzman etc.
A recurrent neural network has (at least one) cyclic path of synaptic connections. Basic characteristics:
1. all biological neural networks are recurrent
2. mathematically, they implement dynamical systems
3. several types of training algorithms are known, no clear winner
4. theoretical and practical difficulties by and large have prevented practical applications so far.

Training of neural network
Once a network has been structured for a particular application, it is ready for training. At the beginning, the initial weights are chosen randomly and then the training or learning begins. There are two approaches to training; supervised and unsupervised.

1-Supervised training
In supervised training, both the inputs and the outputs are provided. The network then processes the inputs and compares its resulting outputs against the desired

outputs. Errors are then propagated back through the system, causing the system to adjust the weights, which control the network. This process occurs over and over as the weights are continually loop. The set of data, which enables the training, is called the "training set." During the training of a network, the same set of data is processed many times, as the connection weights are ever refined. Sometimes a network may never learn. This could be because the input data does not contain the specific information from which the desired output is derived. Networks also don't trained if there is not enough data to enable complete learning, supervised training needs to hold back a set of data to be used to test the system after it has undergone its training. If a network simply can't solve the problem, the designer then has to review the input and outputs, the number of layers, the number of elements per layer, the connections between the layers, the summation, transfer, and training functions, and even the initial weights themselves. Another part of the designer's creativity governs the rules of training. There are many laws (algorithms) used to implement the adaptive feedback required to adjust the weights during training. The most common technique is known as back-propagation.

2-Unsupervised or adaptive training

The other type is the unsupervised training (learning). In this type, the network is provided with inputs but not with desired outputs. The system itself must then decide what features it will use to group the input data. This is often referred to as self-organization or adaption. These networks use no external influences to adjust their weights. Instead, they internally monitor their performance. These networks look for regularities or trends in the input signals, and makes adaptations according to the function of the network. Even without being told whether it's right or wrong, the network still must have some information about how to organize itself. Competition between processing elements could also form a basis for learning. Training of competitive clusters could amplify the responses of specific groups to specific stimuli. As such, it would associate those groups with each other and with a specific appropriate response. Normally, when competition for learning is in effect, only the weights belonging to the winning processing element will be updated. The unsupervised learning is not well understood and there continues to be a lot of research in this aspect.

The Perceptron Training Algorithm

Is the single layer network , it is similar to Pitts neuron . The input values and activation levels of the perceptron are either -1 or 1; weights are real valued. The activation level of the perceptron is given by summing the weighted input values, ∑xiwi, where an activation above a threshold results in an output value of 1, and -1 otherwise. Given input values Xi, weights Wi, and a threshold t, the perceptron computes its output value as:

1 if $\sum xi,wi >= t$

-1 if $\sum xiwi < t$

The perceptron uses a simple form of supervised learning. After attempting to solve a problem instance, a teacher gives it the correct result. The perceptron then changes its weights in order to reduce the error, The following rule is used.

Let c be a constant whose size determines the learning rate and d be the desired output value. The adjustment for the weight on the ith component of the input vector Δwi is given by:

$$\Delta wi = c(d - sign(\sum xiwi))\ xi$$

The sign(∑wixi) is the perceptron output value. It is +1 or -1.

The difference between the desired output and the actual output values will thus be 0, 2, or -2. Therefore for each component of the input vector:

If the desired output and actual output values are equal, do nothing.

If the actual output value is -1 and should be 1, increment the weights on the ith line by 2cxi.

If the actual output value is 1 and should be -1, decrement the weights on the ith line by 2cxi.

This procedure has the effect of producing a set of weights which are intended to minimize the average error over the entire training set. If there exists a set of weights which give the correct output for every member of the training set, the perceptron learning procedure will learn it.

An example: using perceptron network for classify