**ARTIFICIAL NEURAL NETWORK**

**1.1 Introduction to Artificial Neural Network**

An ANN is an information processing system that is based on generalization of human cognition or neural biology. Fausett gives these assumptions between the two:

Information processing occurs at many simple elements called neuron. Signals are passed between neurons over connection links. Connection link has an associated weight, which is a typical neural net, multiplies the signal transmitted. Each neuron applies an activation function (non linear) to its net input to determine its output signal.

ANN is characterized by:

* Architecture – Its pattern of connections between neurons.
* Learning algorithm – Its method of determining the weights on connections.
* Activation function – determines its output.

ANN consists of a large number of processing elements called neurons. Each neuron has an internal state called its activation as a signal to several other neurons. A neuron can send only one signal at a time although that signal may be broadcast to other neurons.

Artificial neural networks are biologically inspired, i.e. they are composed of elements that perform in a manner that is analogous to the most elementary functions of the biological neurons. Despite the functional similarities not even the most optimistic advocate will suggest that ANNs will soon duplicate the function of the human brain.

**1.2 Neural Network**

It is a parallel distributed information processing structure in the form of a directed path or graph. The nodes of the graph is called processing element. The links of the graph is called connections. Each processing element can receive any number of incoming connections. Each processing element can have any number of outgoing connections, but the signals in all of these must be same. Processing elements can have local memory. Each processing element possesses a transfer function which can use local memory, can use input signal and which produces the processing elements output signal. Transfer function can operate continuously or episodically. Processing elements transfer function usually have a sub function, called learning law. The identifier used to describe the group to which a particular input connection belongs-inputclass. All the connections belonging to a particular input class are required to have signals of the same mathematical type.

A fascicle is a collection of connection layers are also known as slabs. A vector in which has weights as components-weight vector. Network weight vector is the vector formed by concatenating all of the weights of all of the individual processing elements of the network.

Weight space: a weight is a local memory variable of a specified mathematical data type that is assigned to each input connection or to specified mathematical combinations of input connection of a particular class.

**1.3 Advantages**

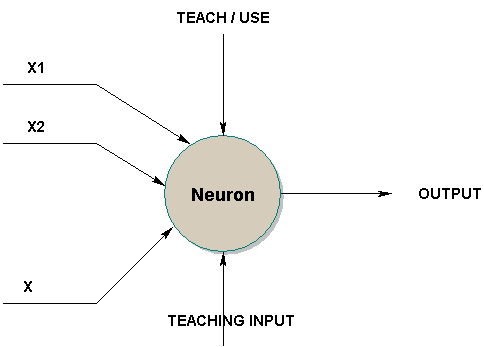
Neural network with their remarkable ability to derive meaning from complicate or imprecise data can be used to extract pattern and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an ‘expert’ in the category of information it has been given to analyse .This expert can then be used to provide projections give new situations of interest and answer “what if” questions. Other advantages include

1. Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.
2. Self organization: An ANN can create its own organization or representation of the information it receives during learning time.
3. Real time operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantages of this capability.
4. Fault tolerance via redundant information coding: partial destruction of a network leads to the corresponding degradation of performance. However some network capabilities may be retained even with major network damage.

**1.4 Structure of Neuron**

**1.4.1 A Simple Neuron**

An artificial neuron is a device with many inputs and one output. The neuron has two modes of operation; the training mode and the using mode. In the training mode, the neuron can be trained to fire( or not) for particular input patterns. In the using mode when a taught input pattern is detected at the input its associated output becomes the current output if the Input pattern does not belong in the taught list of input patterns.



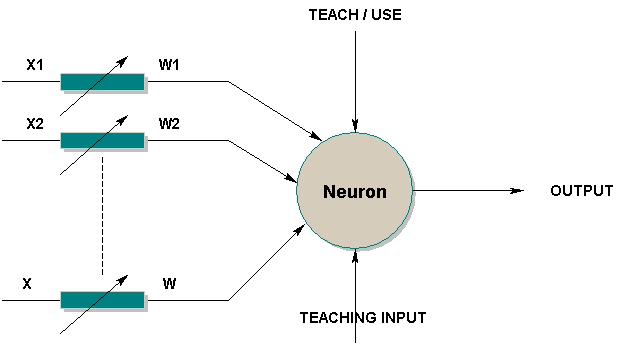
Simple Neuron

**1.4.2 Firing Rules**

The firing rule is an important concept in neural network and accounts for their high flexibility. A firing rule determines how one calculates whether a neuron should fire far any input pattern.

**1.4.3 A more complicated neuron.**

The previous neuron doesn’t do anything that conventional computers don’t do already. A more sophisticated neuron (Fig. 2) is the Meculloch and Pitts model (MCP). The difference from the previous model is that the inputs are weighted, the effect that each input has at decision making is dependent on the weight of the particular input. The weight of an input is a number which when multiplied with the input gives the weighted input. These weighted inputs are then added together and if they exceed a pre-set threshold value, the neuron fires in any other case the neuron does not fire.



In mathematical terms, the neuron fires if and only if

x1w1+x2w2+x3w3+………>T

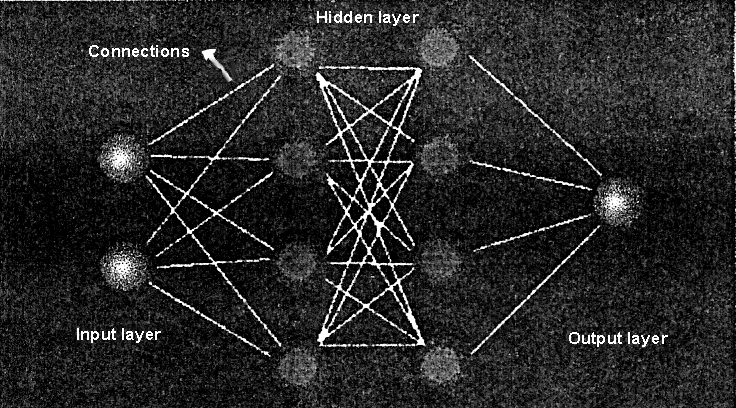
The addition of input weights and of the threshold makes this neuron a very flexible and powerful one. The MCP neuron has the ability to adapt to a particular situation by changing its weights and/or threshold. Various algorithms exist that cause the neuron to ‘adapt’; the most used ones are the Delta rule and the back error propagation. The former is used in feed forward network and the latter is feedbacks networks.

**1.5 Neural Network Architecture**

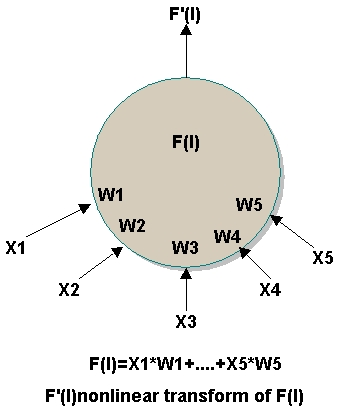
“Architecture of neural neural networks is the set of possible Interconnections (also called as topology of the network) and the learning algorithm defined for it.”

**1.5.1 Layers**

An artificial neural network consist of a large number of processing elements (perceptron). Neural networks are typically organized in layers. Layers are made up of a number of interconnected ‘nodes’ which contain an ‘ activation function’. Patterns are presented to the network via the ‘input layers’ which communicates to one or more ‘hidden layers’ where the actual processing is done via a system of weighted ‘ connections’. The hidden layers then link to an ‘output layer’ where the answer is output.



Each node in the hidden layer is fully connected to the inputs. That means what is learned in a hidden node is based on all the inputs taken together. This hidden layer is where the network learns interdependencies in the model. The following diagram provides some detail into what goes on inside a hidden node.



Simply speaking a weighted sum is performed:

x1 times w1 plus x2 times w2 on through x5 times w5. This weighted sum is performed for each hidden node and each output node and is how interactions are represented in the network. Each summation is then transformed using a nonlinear function before the value is passed on to the next layer. Further more, not all layers must have the same number of neurons. The term feedforward networks defines the topology that the example illustrate all neurons of the input layer are connected to each neurons of the next layer. Further more, no input connects to the neurons of the same or previous layer neurons there is no feedback. The topology is most widely and generally used topology of the ANNs. Biologist agree that there are different kinds of neurons in the body of a human being. Similarly there are a large number of ANNs developed by different researchers. About 70% of the ANNs used now a days is based on the feed forward topology.

**1.6 Design**

The developer must go through a period of trial and error in the design decisions before coming up with a satisfactory design. The design issues in neural networks are complex and are the major concerns of system developers.

Designing a neural network consist of:

* Arranging neurons in various layers.
* Deciding the type of connections among neurons from different layers as well as among the neurons within a layer.
* Deciding the way a neuron receives input and produce output.
* Determining the strength of connection within the network by allowing the network learn the appropriate values of connection weights by using a training data set.

**1.7 Types of Neural Network**

Neural networks can be classified according to their structures and learning algorithms.In terms of their structures neural network can be divided into two ways:

* Feed forward network
* Recurrent networks

**1.7.1 Feedforward networks**

In a feed forward network, the neurons are generally grouped into two layers. Signals flow from the input layer to the output layer via unidirectional connections and neurons being connected from one layer to the next, but not within the same layer Eg MLP, Learning vectors quantization and group method of data handling network.

In feedforward network neurons of one layer are only connected with neurons of the succeeding layer without any recurrent connections. Normally these nets consist of one input layer, one or two hidden layer (called hidden, since they don’t have a direct connection to the outside world) and one output layer. With such a net, input data are mapped from the n-dimensional input space to an n-dimensional output space. This network now has to learn to produce a certain desired output pattern presented at the input layer.

**1.7.2 Recurrent network**

In a recurrent network, the output of some neurons is fed back to the same neurons or to neurons in the preceding layer. Thus signals can flow in both forward and backward directions Eg Hopfield networks, the Elman networks and the Jordan networks.

Learning is performed in the following way. we can distinguish two major categories of nerural network.

* Fixed network: in which the weights cannot be changed, ie dw/dt = 0. In such networks the weights are fixed a prior according to the problem to solve
* Adaptive network: which are able to change their weights ie dw/dt not = 0

**1.7.3 Neural network model**

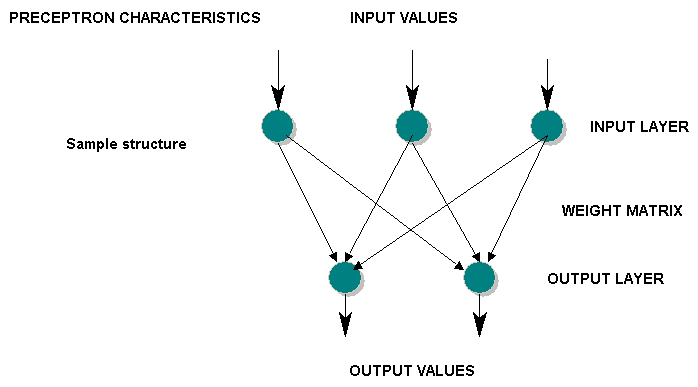
The single artificial neurons can now be interconnected in many different ways leading to a variety of neural networks with different architecture, learning rules and abilities.

The most important ones are: Feedforward network, Adaptive Resonance theory (ART), Hopfield nets, Kohonen’s self organizing feature maps. Radial basic functions(RBF), Boltzmann-machines and cascade-correlation. Simple perceptrons, BP and RBF networks are supervised networks where as Kohonen and Carpenter/Grossberg are unsupervised.

**1.7.4 Single layer perceptron**

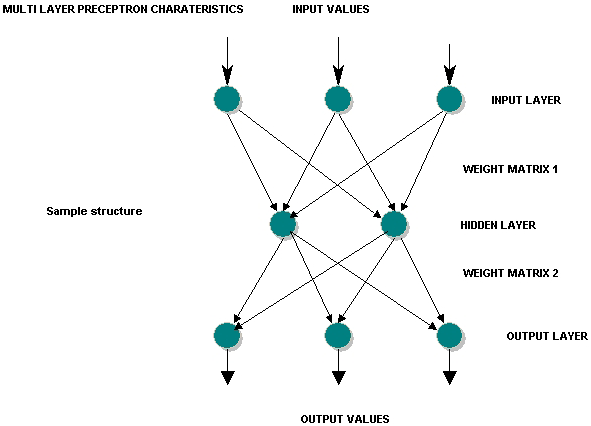
Mc Culloch and Pitts(1943) published the first systematic study of artificial neural network. In later work they explore network paradigm for pattern recognition despite translation and rotation. Much of their work involved the simple neural model. The sum unit multiplies each input X by a weight W and sums the weighted inputs. If this sum is greater than a predetermined threshold , the output is one; otherwise it is zero.

These systems are collectively been called perceptrons. Perceptrons consist of a single layer of artificial neurons connected by weights to set of inputs.



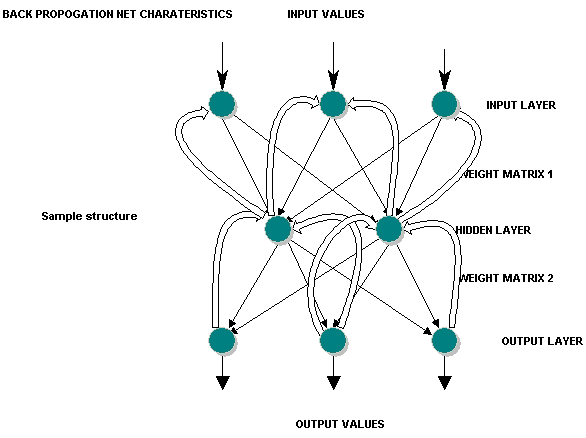
**1.7.5 Multilayer perceptron**

Cascading two single layer networks forms multilayer perceptron (MLP).The multilayer perceptron has one or more hidden neuron layers between its input and output layers.Due to its extended structure, a multilayer perceptron is able to solve every logical operations, including the XOR problem.



**1.7.6 Backpropagation network**

The backpropagation network was first introduced by G.E.Hinton, E.Rumelhart and R.J.Williams in 1986 and is one of the most powerful neural network types. It has the same structure as the multilayer perceptron and uses the backpropagation learning algorithm.



**1.7.7 Hopfield network**

The Hopfield network was first introduced by Physicist J.J.Hopfield in 1982 and belongs to neural net types which are called “thermodynamical models”. It consists of a set of neurons, where each neuron is connected to each other neuron. There is no differenciation between input and output neurons . The main application of a hopefield network is the storage and recognition of patterns Eg: Image files

### A Description of the Hopfield Network

The **Hopfield neural network** is a simple artificial network which is able to store certain memories or patterns in a manner rather similar to the brain - the full pattern can be recovered if the network is presented with only partial information. Furthermore there is a degree of stability in the system - if just a few of the connections between nodes (neurons) are severed, the recalled memory is not too badly corrupted - the network can respond with a "best guess". Of course, a similar phenomenon is observed with the brain - during an average lifetime many neurons will die but we do not suffer a catastrophic loss of individual memories - our brains are quite robust in this respect (by the time we die we may have lost 20 percent of our original neurons).

The nodes in the network are vast simplifications of real neurons - they can only exist in one of two possible **"states"** - **firing** or **not firing**. Every node is connected to every other node with some strength. At any instant of time a node will change its state (i.e start or stop firing) depending on the inputs it receives from the other nodes.

If we start the system off with any general pattern of firing and non-firing nodes then this pattern will in general change with time. To see this think of starting the network with just one firing node. This will send a signal to all the other nodes via its connections so that a short time later some of these other nodes will fire. These new firing nodes will then excite others after a further short time interval and a whole cascade of different firing patterns will occur. One might imagine that the firing pattern of the network would change in a complicated perhaps random way with time. The crucial property of the Hopfield network which renders it useful for simulating memory recall is the following: we are *guaranteed* that the pattern will settle down after a long enough time to some fixed pattern. Certain nodes will be always **"on"** and others **"off"**. Furthermore, it is possible to arrange that these *stable firing patterns* of the network correspond to the desired memories we wish to store!

The reason for this is somewhat technical but we can proceed by analogy. Imagine a ball rolling on some bumpy surface. We imagine the position of the ball at any instant to represent the activity of the nodes in the network. Memories will be represented by special patterns of node activity corresponding to wells in the surface. Thus, if the ball is let go, it will execute some complicated motion but we are certain that eventually it will end up in one of the wells of the surface. We can think of the height of the surface as representing the energy of the ball. We know that the ball will seek to minimize its energy by seeking out the lowest spots on the surface -- the wells.

Furthermore, the well it ends up in will usually be the one it started off closest to. In the language of memory recall, if we start the network off with a pattern of firing which approximates one of the "stable firing patterns" (memories) it will "under its own steam" end up in the nearby well in the energy surface thereby recalling the original perfect memory.

The smart thing about the Hopfield network is that there exists a rather simple way of setting up the connections between nodes in such a way that any desired set of patterns can be made "stable firing patterns". Thus any set of memories can be burned into the network at the beginning. Then if we kick the network off with any old set of node activity we are *guaranteed* that a "memory" will be recalled. Not too surprisingly, the memory that is recalled is the one which is "closest" to the starting pattern. In other words, we can give the network a corrupted image or memory and the network will "all by itself" try to reconstruct the perfect image. Of course, if the input image is sufficiently poor, it may recall the incorrect memory - the network can become "confused" - just like the human brain. We know that when we try to remember someone's telephone number we will sometimes produce the wrong one! Notice also that the network is reasonably robust - if we change a few connection strengths just a little the recalled images are "roughly right". We don't lose any of the images completely.

### Hopfield Network - A Simulation

As a simple example of this, we might design a neural network to store images of the numbers zero through nine represented as patterns on a two-dimensional grid. To be concrete, let there be 256 nodes on a 16-by-16 grid. Think of the nodes as "light bulbs", which can be on or off according to their firing state. First, we must *train* the network to remember these images. To do this, we present the patterns one after another repeatedly to the network, and, by comparing the network output to the target output, we can adjust connections in order that the output of the network resembles the target output more and more closely.

It is often inconvenient to build such an artificial neural computer every time we want to experiment with a new network design or store a new set of memories. Instead what is often done is to create a ***simulation*** of such a network using a conventional computer. To do this we can write a computer program which *emulates* exactly what the true neural computer would do using a specially constructed program. Although this program takes much longer to execute than the neural computer it can be made to produce the same answers. It is then easy to modify this program to store a new set of memory patterns or to do some new task.

**1.7.8 Kohonen feature map**

TheKohonen feature map is probably the most useful neural network types, if the learning process of the human brain shall be simulated. The “heart” of this type is the feature map ,a neuron layer were neurons are organizing themselves according to certain input values. The types of the neural network is both feed forword (input layer to feature map) and feed back(feature map).

The objective of a Kohonen network is to map input vectors (patterns) of arbitrary dimension N onto a discrete map with 1 or 2 dimensions. Patterns close to one another in the input space should be close to one another in the map: they should be topologically ordered. A Kohonen network is composed of a grid of output units and N input units. The input pattern is fed to each output unit. The input lines to each output unit are weighted. These weights are initialised to small random numbers.

### Learning in Kohonen Networks

The learning process is as roughly as follows:

* initialise the weights for each output unit
* loop until weight changes are negligible
  + for each input pattern
    - present the input pattern
    - find the winning output unit
    - find all units in the neighbourhood of the winner
    - update the weight vectors for all those units
  + reduce the size of neighbourboods if required

The winning output unit is simply the unit with the weight vector that has the smallest Euclidean distance to the input pattern. The neighbourhood of a unit is defined as all units within some distance of that unit on the map (not in weight space). In the demonstration below all the neighbourhoods are square. If the size of the neighbourhood is 1 then all units no more than 1 either horizontally or vertically from any unit fall within its neighbourhood. The weights of every unit in the neighbourhood of the winning unit (including the winning unit itself) are updated using

|  |  |
| --- | --- |
| \begin{displaymath}\vec{w}_{i} = \vec{w}_{i} + \alpha \, ( \vec{x}_{i} - \ vec{w}_{i}) \end{displaymath} |  |

This will move each unit in the neighbourhood closer to the input pattern. As time progresses the learning rate and the neighbourhood size are reduced. If the parameters are well chosen the final network should capture the natural clusters in the input data.

# Algorithm

line

Kohonen's self-organizing mapping algorithm   
  
**Step 1**. Each neuron in the Kohonen layer receives a complete copy of an input pattern.   
  
**Step 2**. Find the wining neuron. The winning neuron is the one with the smallest distance dj.

m

\---

dj = [ > (Xi - Wij)2]0.5

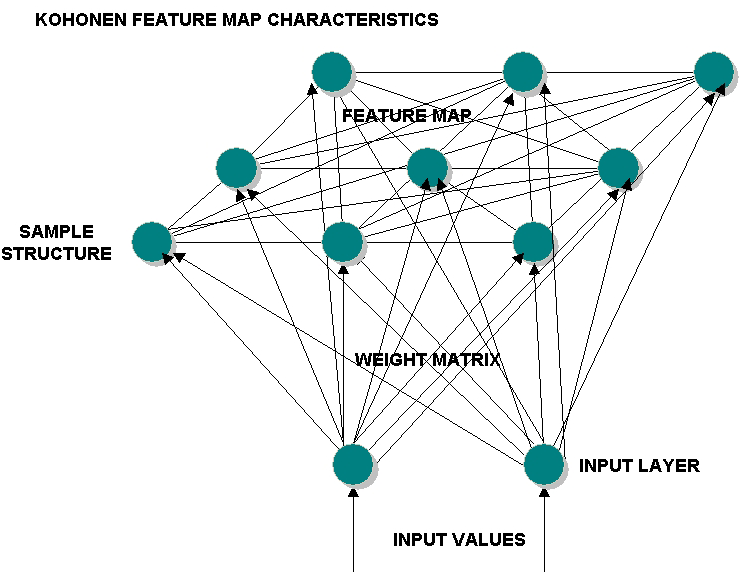
/---

i=1

**Step 3**. For the winning neuron and its physical neighbors, the following training law is used to modify weights.

Wij(t+1)=Wij(t)+alpha(t)\*gamma(t)\*[Xi-Wij(t)]   
gamma(t)=exp{-0.5\*[rij/sigma(t)]2}   
  
where alpha is the learning speed decreasing with time (initial value is between 0 and 1), rij is the distance between the winning cell and the cell being updated (this distance can be calculated with or without edges), and sigma is the neighborhood radius decreasing with time.   
  
**Step4**.Repeat **Steps 1-3** with a new input pattern .

**Step 5**. Repeat **Step 4** until the entire input patterns have been passed through (this constitutesat)   
  
**Step 6**. Repeat **Step 5** for a specified number of times.



**1.8 The Learning Process**

Information is stored in the weight matrix w of a neural network. Learning is the determination of the weights.

The brain basically learns from experience. Neural networks are sometimes called machine learning algorithm, because changing of its connection weights (training) causes the network to learn the solution to a problem. The strength of connection between the neurons is stored as a weight value for the specific connection.The system learns new knowledge by adjusting these connection weights.

The learning ability of a neural network is determined by its architecture and by the algorithmic method chosen for training. The training method usually consists of one of three schemes.

**1.8.1** **Unsupervised learning**

The hidden neurons must find a way to organize themselves without help from the outside. In this approach, no sample outputs are provided to the network against which it can measure its predictive performance for a given vector of inputs. This is learning by doing.

**1.8.2** **Reinforcment learning**

This method works on reinforcement from the outside. The connections among the neurons in the hidden layer are randomly arranged, then reshuffled as the network is told how close it is to solving the problem. Reinforcement learning is also called supervised learning, because it requires a teacher. The teacher may be a training set of data or an observer who grades the performance of the network results.

Both unsupervised and reinforcement suffers from relative slowness and inefficiency relying on a random shuffling to find the proper connection weights.

**1.8.3 Back propagation**

This method is proven highly successful in training of multilayered neural networks. The network is not just given reinforcement for how it is doing on a task. Information about errors is also filtered back through the system and is used to adjust the connections between the layers, thus improving performance.

**1.8.4 Transfer Function**

The behavior of an ANN (Artificial neural network)depends on both the weights and the input-output function(transfer function)that is specified for the units. This function typically falls into one of three categories.

* Linear
* Threshold
* Sigmoid

For linear units, the output activity is proportional to the total weighted output. For threshold units the output is set at one of the two levels, depending on whether the total input is greater than or less than some threshold value.

For sigmoid units, the output varies continuously but not linearly as the input changes. Sigmoid units bear a greater resemblance to real neurons that do linear or threshold units but all three must be considered rough approximations.

To make a neural network that performs some specific task, we must choose how the units are connected to one another and we must set the weights on the connections appropriately. The connection determines whether it is possible for one unit to influence another. The weights specify the strength of the influence.

**2.10 Application of Neural Networks**

Their applications are almost limitless but fall in to a few simple categories. Basically most applications of neural networks fall into the following five categories.

* Prediction

Uses input values to predict some output E.g. pick the best stocks in the market ,predict weather identify people with cancer risk.

* Classification

Uses input values to determine the classification Eg is the input the letter A, is the blob of the video data, a plane and what kind of plane is it.

* Data association

Like classification but it also recognizes data that contains errors Eg not only identify the characters that were scanned but identify when the scanner is not working properly.

* Data conceptualization

Analyze the inputs so that grouping relationship can be inferred Eg extract from a database the name of those most likely to buy a particular product.

* Data filtering

Smooth an input signal Eg take the noise out of a telephone signal.

Neural networks applications in industry, business and science is increasing now a days. The important applications are:

 Direct application

 Pattern classification

 Recognition of Olympic symbols

 Recognition of printed characters

 Making an opening bid in construct budge game

 Associative memories

 Image pattern recall

 Content addressable memory

 Information retrieval

 Optimization

 Graph bipartition problem

 Linear programming problem

 Traveling salesman problem

 Smoothing images with discontinuities

 Vector quantization

 Control application

 Application areas

 Application in speech

 Net talk

 Phonetic typewriter

 Vowel classification

 Recognition of consonant-vowel(cv) segments

 Recognition of stops cv iterance in Indian languages.

 Application in image processing

 Recognition of hand written digits

 Image segmentation

 Texture classification and segmentation

 Application of decision-making

 Prediction and financial analysis

 Loan approval

 Real estate analysis

 Marketing analysis

 Airline seating allocation

 Military application

 Missile guidance and detonation

 Fighter flight and battle pattern guidance

 Optical telescope

 Vehicular trajectory control

 Automatic applications

* Biological application and drug development

Used in research on amino acid sequencing in protein, nucleotide sequencing in RNA and DNA, and ECG waveform classification, prediction of patients reactions to drug treatments, prevention of anesthesia related accidents. They are used in predicting the medical properties of substances without expensive time consuming and often inhuman animal testing.

 Electronic motor prediction

 Mass spectra classification

New application areas:

Pen Pc’s

Pc’s where one can write on a tablet, and the writing will be recognized and translated into (ASCII)text.

Speech and vision recognition systems

Not new, but neural networks are becoming increasingly part of such systems. They are used as a system component, in conjunction with traditional computers.

White goods and toys

As neural network chips become available, the possibility of simple cheap systems which have learned to recognize simple entities Eg walls looming or simple commands like (Go or Stop), may lead to their incorporation in toys and washing.

**2. ADAPTIVE RESONANCE THEORY**

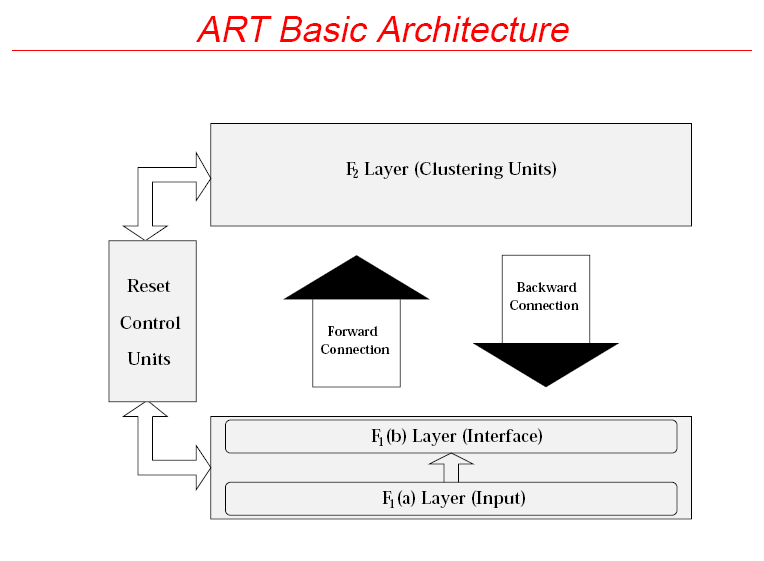
**2.1 Introduction to Adaptive Resonance Theory**

Adaptive resonance theory (ART) was proposed by Grossberg in 1970s in order to analyze human cognitive information processing.Its basic computational design goals have always included memory stability with fast or slow learning in open and evolving input environments. A serial of ART neural network models, by Carpenter *et al*;have realised the above principles as quantitative systems and further added new principles to the original theory. There is a well-known dilemma, stability-plasticity, in neural network community. How can a learning system be designed to remain plastic or adaptive in response to significant events and yet remain stable in response to irrelevant events? The ART networks are capable of solving this dilemma, while many neural network models suffer such a dilemma.

**2.2 ART Motivation and Features**

To be able to mimic biological behaviour, the emphasis of ART networks lies at unsupervised learning and self-organization. The self-organization process of unsupervised learning leads to a pattern clustering or a category learning procedure. ”Resonance” refers to the so-called resonant state of the network in which a category prototype vector matches the current input vector close enough. ART networks are designed to allow a user to control the degree of similarity of patterns placed on the same cluster. To overcome the stability-plasticity dilemma, a bi-directional structure is adopted and controlled by a set of rather intricate operations. There appear several simplified versions of ART networks, where complicated operations have been ruled out and only clustering properties remain.

**2.3 ART Basic Architecture**



* Three groups of neurons: an input processing field (F1 layer), the cluster units (F2 layer), and a mechanism for control and reset.
* F1 layer can be considered to consist of two parts: the input and the interface portions, F1(a) and F1(b).
* To control similarity of patterns placed on the same cluster, there are two sets of connections, top-down and bottom-up, between each unit in F1 (b) and each cluster unit in F2.
* F1 (b) combines input signals from F1 (a) and F2 layer to measure similarity between an input signal and the weight vector for the specific cluster unit.
* The F2 layer is a competitive layer: the cluster unit with the largest activation becomes the candidate to learn the input pattern. (Winner-take-all)
* Whether or not this cluster unit is allowed to learn the input pattern depends on how similar its top-down weight is to the input vector.
* The decision is made by the reset unit, and other supplemental units are also needed to control the processing of information in the ART networks.

**2.4 A general operational procedure of ART**

0. Initialize parameters.

1. While the stopping condition is NOT satisfied, do Steps 2-9.

2. For each input vector, do Steps 3-8.

3. Process F1 layer.

4. While reset condition is true, do Steps 5-7.

5. Find a candidate unit to learn the current input pattern: F2

6. F1 (b) units combine their inputs from F1(a)and F2.

7. Test reset condition:

If reset is true, the current candidate unit is rejected (inhibited); go to Step 4. Otherwise, the current candidate unit is accepted for learning; proceed Step 8.

8. Learning: weight changes according to differential equations.

9. Test stopping condition.

**2.5 General comments on ART basic operations**

At any time, an F2 unit is in one of three states as follows.

- active: “on”, activation = d (For different ART networks, d takes a different value).

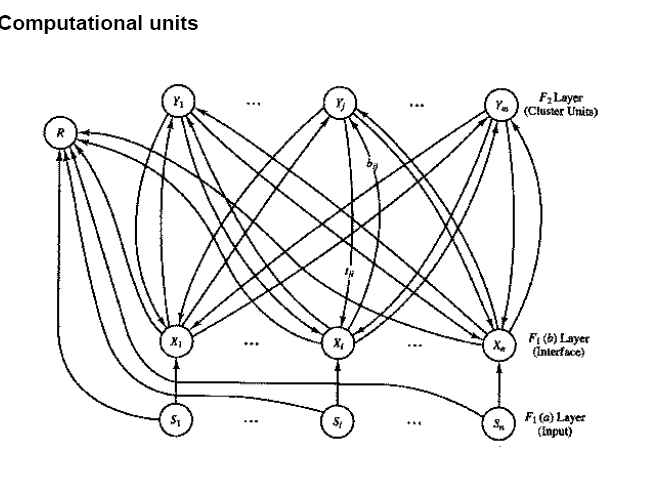
- inactive: “off”, activation = 0, but available to participate in competition.

- inhibited: “off”, activation = 0, and prevented from participating in any further competition during the presentation of the current input vector. A learning trial consists of the presentation of one input pattern. The calculation in Step 2 constitutes a learning trial. Prior to the presentation of the pattern, the activations of all units are set to zero but all F2 units are inactive. The similarity degree for assigning a pattern to the same cluster unit is controlled by the vigilance parameter, a user-tunable parameter. Once an acceptable cluster unit has been selected for learning, the bottom-up and top-down signals are maintained for an extended period, during which time the weight changes occur (resonance).

**2.6 Overview**

ART-1 is designed to cluster binary input vectors (nonzero) and direct user control of the degree of similarity among patterns placed on a cluster unit. The architecture of ART-1 network consists of computational and supplemental units. The learning process is designed such that patterns are not necessarily presented in a fixed order and the number of patterns for clustering may be unknown in advance. Updates for both the bottom-up and top-down weights are controlled by differential equations. However, this process may be finished in a learning trial. In the other words, the weights reach the equilibrium during each learning trial.

**3.7 Computational units**



* The architecture of the computational units consists of F1 units (input and interface units), F2 (cluster units), and a reset unit that carries out user control over the degree of similarity.
* Connections:

- Each unit in F1(a) layer is connected to the corresponding unit in F1(b).

- Each unit in F1(b) is connected to each unit in F2 by two weighted pathways.

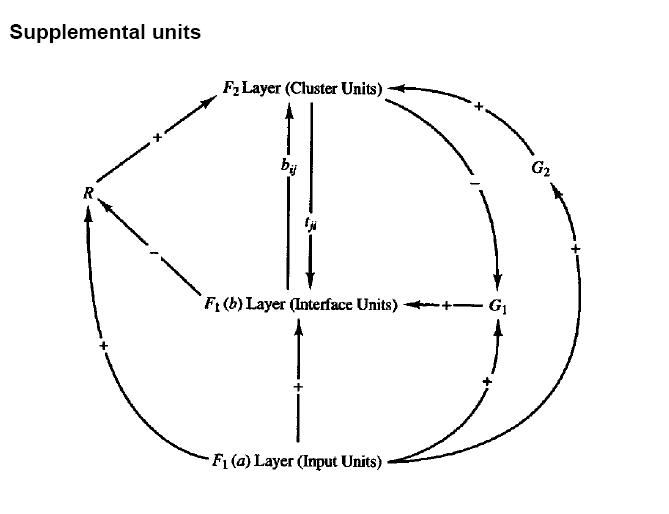
- tji: the top-down weight from unit Yj in F2 to unit Xi in F1(b).

- tij: the bottom-up weight from unit Xi in F1(b) to unit Yj in F2.

- The F2 layer is a competitive layer where only the uninhibited node with the largest activation has a non zero value.

- The reset unit. Controls the vigilance matching.

**3.8 Supplemental units**



* From a theoretical viewpoint, the supplemental units are important; they provide a mechanism to perform computation using the neural network principle.
* Units in the ART-1 network need to respond differently at different stages of the process. But a biological neuron does not have a method to decide what to do and when. So does the implementation of a reset mechanism.
* Gain control units, G1 and G2, and the reset unit, R, are introduced to solve the above problem. + and - express the excitatory and the inhibitory signals.
* A signal is sent whenever any unit in the designated layer is “on”.
* “2/3” rule: each unit in either F1(b) or F2 has three possible sources receiving signals. Such a unit is “on” if and only if it receives two excitatory signals.

**Where:**

n : number of components in the input vector.

m : maximum number of clusters that can be formed.

bij : bottom-up weights (from F1(b) unit Xi to F2 unit Yj).

tji : top-down weights (from F2 units Yj to F1(b) unit Xi).

ρ : Vigilance parameter

s : binary input vector (an n-tuple).

x : activation vector for F1(b) layer (binary).

||x|| : norm of vector x, defined as the sum of the binary components xi .

**3.9 Learning algorithm**

Step 0. Initialise parameters: L > 1(constant), 0< ρ ≤ 1).

Initialise weights: 0 < bij(0) < L/(L-1+n) tji=1.

Step 1. While stopping condition is false, do Steps 2-13.

Step 2. For each training input do Steps 3-12.

Step 3. Set activations of all F2 units to zero.

Set activations of all F1(a) units to input vector s.

Step 4. Compute the norm of s: ||s|| = ΣI si.

Step 5. Send input signal from F1(a) to F1(b) layer: xi=si.

Step 6. For each F2 node that is not inhibited:

If yj ≠ -1, then yi = Σi bij xi.

Step 7. While reset is true, do Steps 8-11.

Step 8. Find J such that y J ≥ y j for all nodes J.

If yj= -1, then all nodes are inhibited

(This pattern cannot be clustered.)

Step 9. Recompute activation x of F1(b): xi = si t Ji.

Step 10. Compute the norm of vector x: ||X|| = Σi xi.

Step 11. Test for reset:

If ||x|| / ||s|| < ρ, then yJ = -1(inhibit node J ) and go to Step 7

If ||x|| / ||s|| ≥ ρ, then proceed to Step 12.

Step 12. Update the weights for node J:

biJ(new) = Lxi / L-1+||x||

tJi(new) = xi..

Step 13. Test for stopping condition.

**Comments**

* The zero input is prohibited; i.e.si € {0,1} and Σi si > 0.
* Step 3 removes all inhibitions from the previous learning trial.
* In Step 6, setting y= -1 for an inhibited node will prevent that node from being a winner. (Since all weights and signals in the net are non-negative a unit with a negative activation can never have the largest activation.)
* In Step 8 (winner-take-all), take J to be the smallest such index in case of a tie.
* In Step 9, unit Xi is “on” only if it receives both an external signals si and a signal sent down from F2 to F1, t ji.
* In Step 13, a stopping condition might be one of the following:

- No weight changes.

- No unit reset.

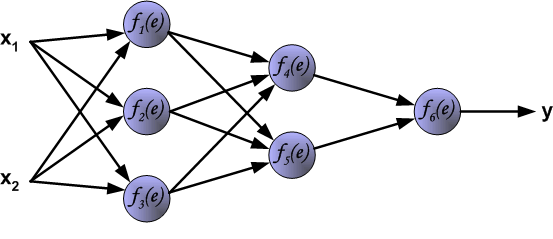
- Maximum number of epochs reached.

References:

An introduction to artificial neural networks James A. Anderson (1995).

# Principles of training multi-layer neural network using *backpropagation* algorithm

Here we describe teaching process of multi-layer neural network employing *backpropagation* algorithm. To illustrate this process the three layer neural network with two inputs and one output,which is shown in the picture below, is used:

****

## The Backpropagation Algorithm

1.

Propagates inputs forward in the usual way, i.e.

* All outputs are computed using sigmoid thresholding of the inner product of the corresponding weight and input vectors.
* All outputs at stage *n* are connected to all the inputs at stage *n*+1

2.

Propagates the errors backwards by apportioning them to each unit according to the

amount of this error the unit is responsible for.

We now derive the stochastic Backpropagation algorithm for the general case. The derivation is simple, but unfortunately the book-keeping is a little messy.

* $\vec{x_j} = $input vector for unit *j* (*xji* = *i*th input to the *j*th unit)
* $\vec{w_j} =$weight vector for unit *j* (*wji* = weight on *xji*)
* $z_j = \vec{w_j}\cdot \vec{x_j}$, the weighted sum of inputs for unit *j*
* *oj* = output of unit *j* ( $o_j = \sigma(z_j)$)
* *tj* = target for unit *j*
* *Downstream*(*j*) = set of units whose immediate inputs include the output of *j*
* *Outputs* = set of output units in the final layer

Since we update after each training example, we can simplify the notation somewhat by imagining that the training set consists of exactly one example and so the error can simply be denoted by *E*.

We want to calculate $\frac{\partial E}{\partial w_{ji}}$for each input weight *wji* for each output unit *j*. Note first that since *zj* is a function of *wji* regardless of where in the network unit *j* is located,

\begin{eqnarray*}\frac{\partial E}{\partial w_{ji}} &=& \frac{\partial E}{\parti...
...rtial w_{ji}} \\
&=& \frac{\partial E}{\partial z_j} x_{ji}\\
\end{eqnarray*}

Furthermore, $\frac{\partial E}{\partial z_j}$is the same regardless of which input weight of unit *j* we are trying to update. So we denote this quantity by $\delta_j$.

Consider the case when $j \in Outputs$ . We know

\begin{displaymath}E = \frac{1}{2}\sum_{k \in Outputs} (t_k - \sigma(z_k))^2
\end{displaymath}

Since the outputs of all units $k \ne j$are independent of *wji*, we can drop the summation and consider just the contribution to *E* by *j*.

\begin{eqnarray*}\delta_j = \frac{\partial E}{\partial z_j} &=& \frac{\partial }...
...o_j)(1-\sigma(z_j))\sigma(z_j)\\
&=& -(t_j - o_j)(1-o_j)o_j\\
\end{eqnarray*}

Thus

|  |  |
| --- | --- |
| \begin{displaymath} \Delta w_{ji} = -\eta \frac{\partial E}{\partial w_ij} = \eta \delta_j x_{ji} \end{displaymath} |  |

Now consider the case when *j* is a hidden unit. Like before, we make the following two important observations.

1.

For each unit *k* downstream from *j*, *zk* is a function of *zj*

2.

The contribution to error by all units $l \ne j$in the same layer as *j* is independent of *wji*

We want to calculate $\frac{\partial E}{\partial w_{ji}}$for each input weight *wji* for each hidden unit *j*. Note that *wji*

influences just *zj* which influences *oj* which influences $z_k \forall k \in
Downstream(j)$each of which influence *E*. So we can write

\begin{eqnarray*}\frac{\partial E}{\partial w_{ji}} &=& \sum_{k \in Downstream(j...
...al o_j} \cdot
\frac{\partial o_j}{\partial z_j} \cdot x_{ji}\\
\end{eqnarray*}

Again note that all the terms except *xji* in the above product are the same regardless of which input weight of unit *j* we are trying to update. Like before, we denote this common quantity by $\delta_j$. Also note that$\frac{\partial E}{\partial z_k} = \delta_k$, $\frac{\partial z_k}{\partial o_j} =
w_{kj}$and $\frac{\partial o_j}{\partial z_j} = o_j (1-o_j)$. Substituting,

\begin{eqnarray*}\delta_j &=& \sum_{k \in Downstream(j)}
\frac{\partial E}{\par...
...
&=& \sum_{k \in Downstream(j)} \delta_k w_{kj} o_j (1-o_j)\\
\end{eqnarray*}

Thus,

|  |  |
| --- | --- |
| \begin{displaymath} \delta_k = o_j (1-o_j) \sum_{k \in Downstream(j)} \delta_k w_{kj} \end{displaymath} | (18) |

We are now in a position to state the Backpropagation algorithm formally.

**Formal statement of the algorithm:**

Stochastic Backpropagation(training examples, $\eta$, *ni*, *nh*, *no*)

Each training example is of the form $\langle \vec{x}, \vec{t}
\rangle$where $\vec{x}$is the input vector and $\vec{t}$is the target vector. $\eta$is the learning rate (e.g., .05). *ni*, *nh* and *no* are the number of input, hidden and output nodes respectively. Input from unit *i* to unit *j* is denoted *xji* and its weight is denoted by *wji*.

* Create a feed-forward network with *ni* inputs, *nh* hidden units, and *no* output units.
* Initialize all the weights to small random values (e.g., between -.05 and .05)
* Until termination condition is met, Do
  + For each training example $\langle \vec{x}, \vec{t}
    \rangle$, Do

1.

Input the instance $\vec{x}$and compute the output *ou* of every unit.

2.

For each output unit *k*, calculate

\begin{displaymath}\delta_k = o_k(1-o_k)(t_k - o_k)
\end{displaymath}

3.

For each hidden unit *h*, calculate

\begin{displaymath}\delta_h = o_h(1-o_h)\sum_{k \in Downstream(h)} w_{kh}\delta_k
\end{displaymath}

4.

Update each network weight *wji* as follows:

\begin{eqnarray*}w_{ji} &\leftarrow& w_{ji} + \Delta w_{ji}\\
{\rm where} \quad \Delta w_{ji} &=& \eta \delta_j x_ji\\
\end{eqnarray*}