Department of Computer Sciences,
University of Kashmir

Artificial Intelligence

Course Code: CSE20515

UNIT-I

1. Introduction to Biological Neural Network

Human brain consists of billions of neurons which is the fundamental processing element of biological neural network. Our brain can be considered as a highly complex, nonlinear and parallel information-processing system. Information is stored and processed in a neural network simultaneously throughout the whole network, rather than at specific locations. In other words, in neural networks, both data and its processing are global rather than local.

The biological neuron (see Figure 1.1) is a nerve cell that provides the fundamental functional unit for the nervous systems. Neurons exist to communicate with one another, and pass electro-chemical impulses across synapses, from one cell to the next, as long as the impulse is strong enough to activate the release of chemicals across a synaptic cleft. The strength of the impulse must surpass a minimum threshold or chemicals will not be released.

Figure 1.1 presents the major parts of the nerve cell:

• Soma

• Dendrites

• Axons

• Synapses

The neuron is made up of a nerve cell consisting of a soma (cell body) that has many dendrites but only one axon. The single axon can branch hundreds of times, however. Dendrites are thin structures that arise from the main cell body. Axons are nerve fibers with a special cellular extension that comes from the cell body.

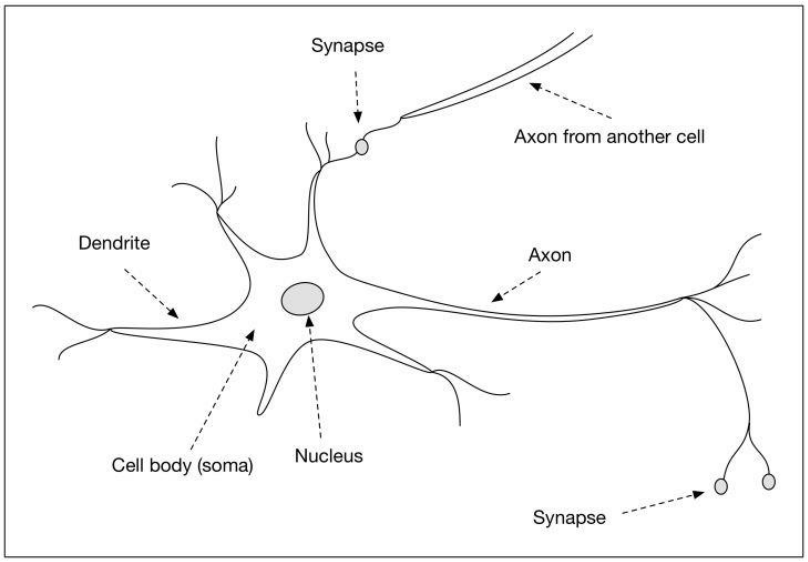


Figure 1.1.The Biological Neuron

Synapses

Synapses are the connecting junction between axon and dendrites. The majority of synapses send signals from the axon of a neuron to the dendrite of another neuron. The exceptions for this case are when a neuron might lack dendrites, or a neuron lacks an axon, or a synapse, which connects an axon to another axon.

Dendrites

Dendrites have fibers branching out from the soma in a bushy network around the nerve cell. Dendrites allow the cell to receive signals from connected neighboring neurons and each dendrite is able to perform multiplication by that dendrite's weight value. Here multiplication means an increase or decrease in the ratio of synaptic neurotransmitters to signal chemicals introduced into the dendrite.

Axons

Axons are the single, long fibers extending from the main soma. They stretch out longer distances than dendrites and measure generally 1 centimeter in length (100 times the diameter of the soma). Eventually, the axon will branch and connect to other dendrites. Neurons are able to send

electrochemical pulses through cross- membrane voltage changes generating action potential. This signal travels along the cell's axon and activates synaptic connections with other neurons.

Information flow across the biological neuron

Synapses that increase the potential are considered excitatory, and those that decrease the potential are considered inhibitory. Plasticity refers the long-term changes in strength of connections in response to input stimulus. Neurons also have been shown to form new connections over time and even migrate. These combined mechanics of connection change drive the learning process in the biological brain. Learning is a fundamental and essential characteristic of biological neural networks. The ease and naturalness with which they can learn led to attempts to emulate a biological neural network in a computer.

2. Artificial Neural Networks

Artificial Neural Network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects;

- Knowledge is acquired by the network through a learning process
- Interneuron connection strengths known as synaptic weights are used to store the knowledge.

ANN can also be defined as a computational or mathematical model composed of a large number of simple, highly interconnected processing elements capable of learning, information processing and problem solving based upon the connectionist approach to computation. Although a present-day artificial neural network resembles the human brain much as a paper plane resembles a supersonic jet, it is a big step forward. ANNs are capable of 'learning', that is, they use experience to improve their performance. When exposed to a sufficient number of samples, ANNs can generalise to others they have not yet encountered. They can recognise hand- written characters, identify words in human speech, and detect explosives at airports. Moreover, ANNs can observe patterns that human experts fail to recognise. For example, Chase Manhattan Bank used a neural network to examine an array of information about the use of stolen credit cards – and discovered

that the most suspicious sales were for women's shoes costing between $40 and $80. Neural nets can be applied to a wide variety of problems, such as storing and recalling data or patterns, classifying patterns, performing general mappings from input patterns to output patterns, grouping similar patterns, or finding solutions to constrained optimization problems.

ANN have become a technical folk legend. The market is flooded with new, increasingly technical software and hardware products, and many more sure to come. Among the most popular hardware implementations are Hopfield, Multilayer Perceptron, Self-Organizing Feature Map, Learning Vector Quantization, Radial basis Function, Cellular Neural and Adaptive Resonance Theory (ART)networks, Counter Propagation Networks, Backpropagation Networks, Neo-cognitron etc. As a result of the existence of all these networks, the application of neural networks is increasing tremendously.

Artificial neural networks have been developed as generalizations of mathematical models of human cognition or neural biology, based on the assumptions that:

- Information processing occurs at many simple elements called neurons.
- Signals are passed between neurons over connection links.
- Each connection link has an associated weight, which, in a typical neural net, multiplies the signal transmitted.
- Each neuron applies an activation function (usually nonlinear) to its net input (sum of weighted input signals) to determine its output signal.

A neural network is characterized by (1) its pattern of connections between the neurons (called its architecture), (2) its method of determining the weights on the connections (called its training, or learning, algorithm), and (3) its activation function.

Since what distinguishes (artificial) neural networks from other approaches to information processing provides an introduction to both how and when to use neural networks, we can define the characteristics of neural networks further as:

A neural net consists of a large number of simple processing elements called neurons, units, cells, or nodes. Each neuron is connected to other neurons by means of directed communication links, each with an associated weight. The weights represent information being used by the net to solve

a problem. Each neuron receives a number of input signals through its connections; however, it never produces more than a single output signal. The output signal is transmitted through the neuron's outgoing connection (corresponding to the biological axon). The outgoing connection, in turn, splits into a number of branches that transmit the same signal (the signal is not divided among these branches in any way). The outgoing branches terminate at the incoming connections of other neurons in the network. Figure 1.2 represents connections of a typical ANN, Typically, a neuron sends its activation as a signal to several other neurons. It is important to note that a neuron can send only one signal at a time, although that signal is broadcast to several other neurons. Weights are the basic means of long-term memory in ANNs. They express the strength, or in other words importance, of each neuron input. A neural network 'learns' through repeated adjustments of these weights. They express the strength, or in other words importance, of each neuron input. A neural network 'learns' through repeated adjustments of these weights.

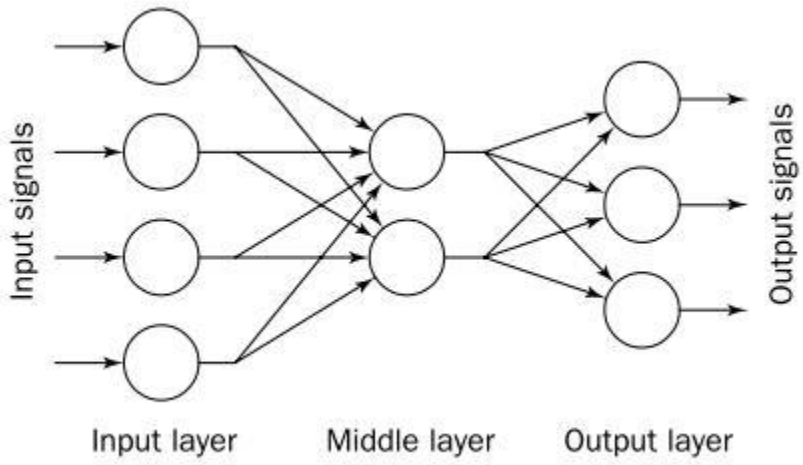


Figure 1.2: Architecture of typical artificial neural network

How neural network adjusts its weights

As shown in Figure 1.2, a typical ANN is made up of a hierarchy of layers, and the neurons in the networks are arranged along these layers. The neurons connected to the external environment form input and output layers. The weights are modified to bring the network input/output behaviour into

line with that of the environment. Each neuron is an elementary information-processing unit. It has a means of computing its activation level given the inputs and numerical weights.

To build an artificial neural network, we must decide first how many neurons are to be used and how the neurons are to be connected to form a network. In other words, we must first choose the network architecture. Then we decide which learning algorithm to use. And finally we train the neural network, that is, we initialise the weights of the network and update the weights from a set of training examples.

3. Analogy between Artificial and Biological Neural Networks

A biological neural network consists of billions of neurons which is the fundamental processing element of biological neural network. a biological neuron consists of a cell nucleus which receives input from other neurons through a web of input terminals, or branches called dendrites. The combination of dendrites is often referred to as a dendritic tree, which receives excitatory or inhibitory signals from other neurons via an electrochemical exchange of neurotransmitters. Depending on this signal aggregated from all synapses from the dendritic tree, the neuron is either activated or inhibited, or in other words, switched on or switched off respectively, after a process called neural summation. The neuron has an electrochemical threshold, analogous to an activation function in artificial neural networks, which governs whether the accumulated information is enough to activate the neuron. The final result is then fed into other neurons and the process begins again. When humans meet new people or see new things every day, they learn what they look like and how they evolve with time. This is achieved by making minor alterations to the neural networks residing in their brains as they evolve. The same phenomenon applies to other tasks as well, both passive sensory tasks, from recognizing generic objects to processing sound as speech patterns, to active task like movement. These skills are learned gradually, and progressively smaller tweaks are used to refine them. The precise topologies are a function of the types of stimuli upon which these biological neural networks are trained.

An artificial neural network consists of a number of very simple and highly interconnected processors, also called neurons. The neurons are connected by weighted links passing signals from one neuron to another. Each neuron receives a number of input signals through its connections; however, it never produces more than a single output signal. The output signal is transmitted

through the neuron's outgoing connection (corresponding to the biological axon). In a nutshell, both types of neuron integrate the signal arriving from multiple other neurons, transform it with a non-linear function and output it to further neurons.

The tasks accomplished by ANNs resemble those accomplished by biological intelligence to some degree, and so do the building blocks of these two types of intelligent systems. Table 1.1 shows the analogy between biological and artificial neural networks.

Table 1.1 Analogy between biological and artificial neural networks

Biological neural network	Artificial neural network
Soma	Neuron
Dendrite	Input
Axon	Output
Synapse	Weight

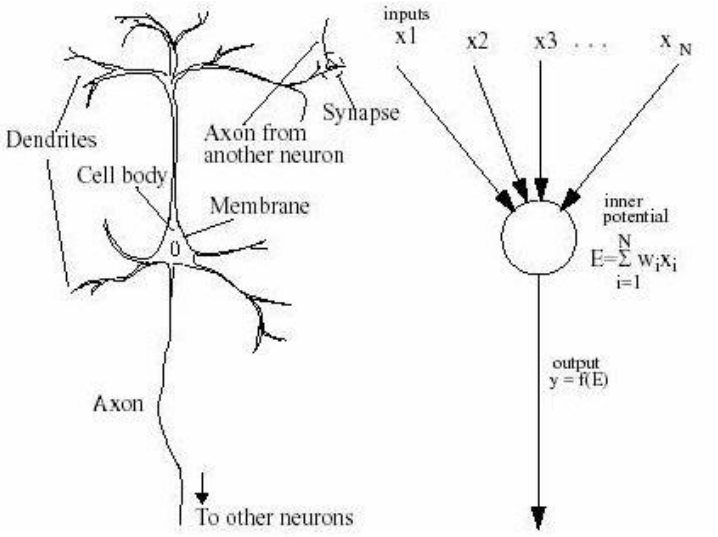


Figure 1.3: Biological Vs Artificial Neuron

4. The Basics of Neuron

The fundamental processing unit of ANN is neuron or node or unit. It is inspired by the biological neuron and performs functions that resemble the biological neurons. ANN's can be made of one to as many as hundreds of neurons depending upon the application in which they would be used. There are three basic elements of neuron: *synapses or connecting links*, *a summing junction* for summing the input signals and *activation function* to ensure that the output of neuron is bounded. A neuron receives several signals from its input links, computes a new activation level and sends it as an output signal through the output links. The input signal can be raw data or outputs of other neurons. The output signal can be either a final solution to the problem or an input to other neurons. Figure 1.4 shows a typical neuron.

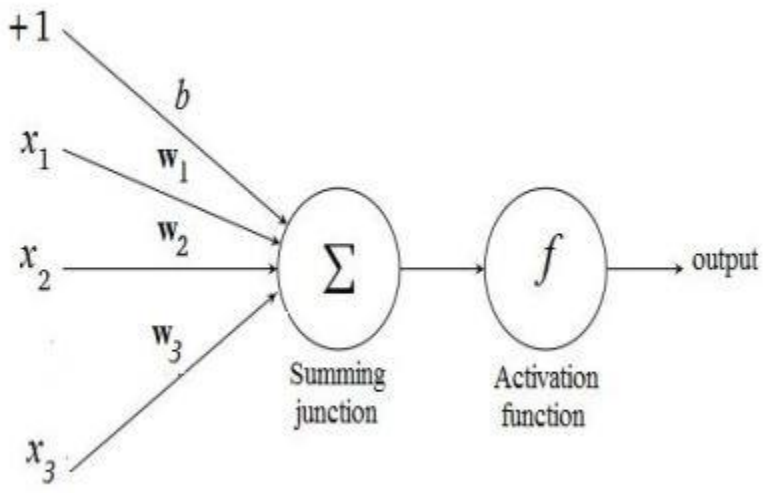


Figure 1.4: Basic Model of Neuron

Each neuron can receive a set of n inputs represented as $x_1, x_2 \ldots . x_N$. Each neuron input has a corresponding synaptic weight $w_1, w_2 \ldots . w_N$. The summation junction performs summation of the signals it receives, with each input signal being multiplied by its associated weights $w_1, w_2 \ldots . w_n$ on the connection and adding the result with the externally applied bias b, in order to enhance the performance of the network. The bias can be thought of as a weight for an input of

unity. The output of summing junction is equal to the sum of the bias and all of the weighted inputs.

$X = \sum_{i=1} w_i x_i + b$ (1.1)

The output of the summing junction is then passed through activation function in order to obtain output y.

$y = f(X)$ (1.2)

$where\ f(.)$ is the Activation function

5. Activation Functions

There are several different kinds of activation functions, each of which is advantageous in different situations. The purpose of activation functions is to ensure that the output of neuron is bounded. That is, the actual response of the neuron is conditioned or damped, as a result of large or small activating stimuli and is thus controllable. In the biological neurons, conditioning of output response is also done, so this concept is uniform with the biological neuron. There are linear as well as non linear activation functions. The similarity between all activation functions is that they ensure that the output of the neuron is restricted to values with a maximum value of 1 and a minimum value of either 0 or -1 depending on the application. The various activation functions are:

(i) Linear Function

The simplest activation function is the linear activation function, where the output signal is directly proportional to the neuron weighted input (Figure 1.5). Neurons which use linear activation function are used for linear approximation as linear functions don't introduce any non linearity

This function is given by:

$y = f(X) = X$ (1.3)

X is the net weighted input

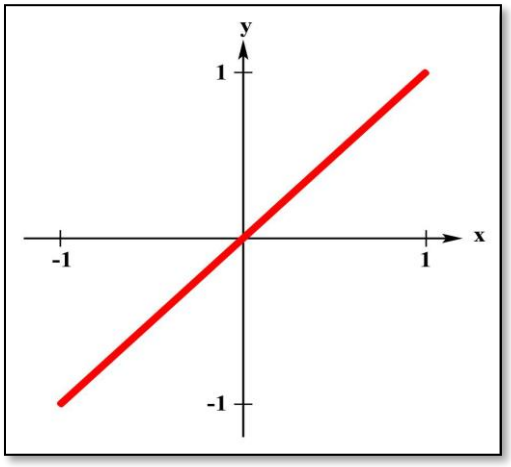


Figure 1.5: The Linear Function

(ii) Step Function

The step function maps the weighted sum of inputs into 0 and 1 that is, a neuron outputs only one of the two possible values as shown in Figure 1.6. The output is zero if the weighted sum of inputs is less than 0 and one otherwise. The step function is not differentiable and thus cannot be used in gradient based training of neural networks such as Backpropagation networks which makes use of activation function derivatives. The step function is also called as *threshold function* or Heaviside function.

The step function is given by

$$y = f(X) = \begin{cases} 1 & if\ X \geq 0 \\ 0 & if\ X < 0 \end{cases} \tag{1.4}$$

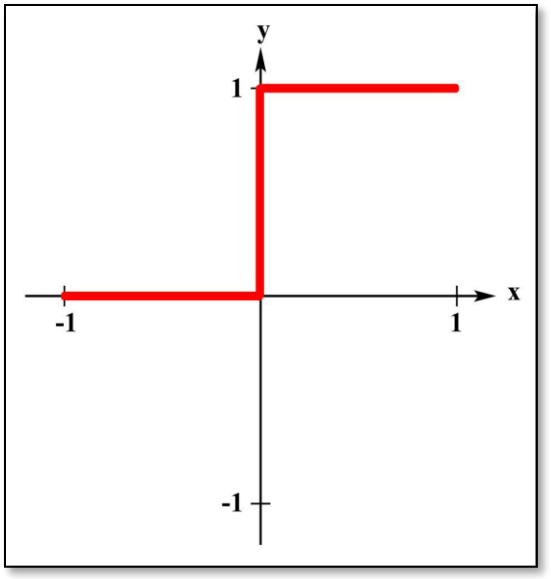


Figure 1.6 Step Function

(iv) Sigmoid function

This type of activation function has an S-shaped graph and is the most common form of activation function used to construct ANN. In contrast to other activation functions, the sigmoid function is both bounded and continuous with a range of value from 0 to 1 as shown in Figure 1.7.The sigmoid function transforms the input which can have any value between$(+\infty, -\infty)$ into a reasonable value in the range between 0 and 1. Sigmoid units bear a greater resemblance to biological neurons than the threshold or linear units and are generally used in Backpropagation networks.

The sigmoid function is given by

$$y = f(X) = \frac{1}{1+e^{-cX}} \qquad (1.5)$$

.

C is the steepness of activation function, which can be varied to decrease the training time and error depending on network architecture and neural network application.

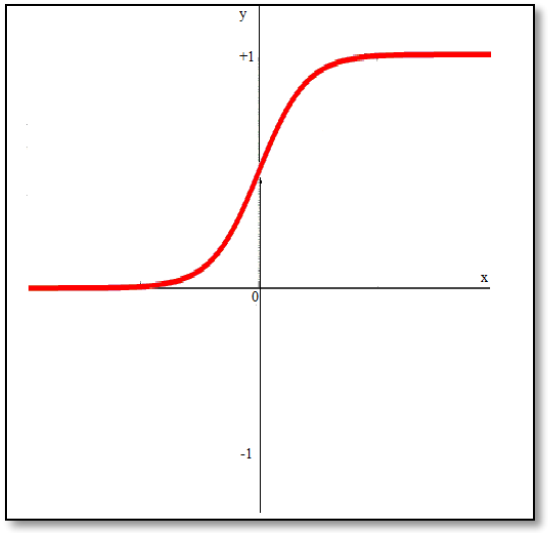


Figure 1.7: Sigmoid Function

(v) Hyperbolic Tangent Function

Hyperbolic tangent function is similar to sigmoid function except that the function range is from -1 to +1 as shown in Figure 1.8. The hyperbolic tangent function is given by

$$y = f(X) = \tanh(X) \tag{1.6}$$

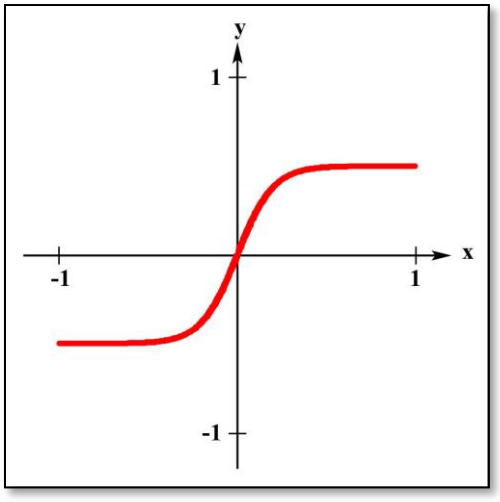


Figure 1.8 Hyperbolic Tangent Function

The choice of the activation function plays a crucial role in ANN performance and therefore cannot be arbitrarily selected.

6. Perceptron

The first mathematical model of biological neuron was introduced by Warren S. McCulloch, a neuroscientist, and Walter Pitts, a logician, known as McCulloch-Pitts model (also known as threshold logic gate) in 1943, it is the simplest model with no learning or adaptation. A McCulloch-Pitts model fires if the sum of excitatory input exceeds the threshold, as long as it doesn't receive an inhibitory input i.e. if the net effect of the external input is greater than a threshold value, the neuron goes into the excited state (gives a response 1) else it remains in the inactive state (gives response 0), often threshold is simply set to 0. McCulloch-Pitts model of neuron has been the basic building block that inspired and stimulated further research in developing paradigm. However this model is so simple that it generates only binary output and also the weights and threshold values are fixed. McCulloch-Pitts model of neuron implements the basic logic operations such as AND, OR, NOT simply by proper choice of weights.

The main purpose of McCulloch-Pitts model of neuron is to model a single neuron. However in this model there exists no mechanism to compare the desired and target outputs i.e. no weight adjustment or learning takes place. The McCulloch-Pitts model attracted the interest of many researchers who build on it and developed the next generation paradigm capable of learning and adaptation. Such a model is a Perceptron, which was introduced by Rosenblatt in early 1960's. The perceptron is the simplest form of a neural network. It consists of a single neuron with adjustable synaptic weights and a hard limiter. A single-layer two-input perceptron is shown in Figure 1.9. There always exists a common bias of '1'. The input neurons are connected to the output neurons through weighted interconnections. This is a single layer network because it has only one layer of interconnections between input and output neurons. The weighted sum of the inputs is applied to the hard limiter (The step and sign activation functions, also called hard limit functions), which produces an output equal to $+1$ if its input is greater than or equal to threshold and 0 if it is less.

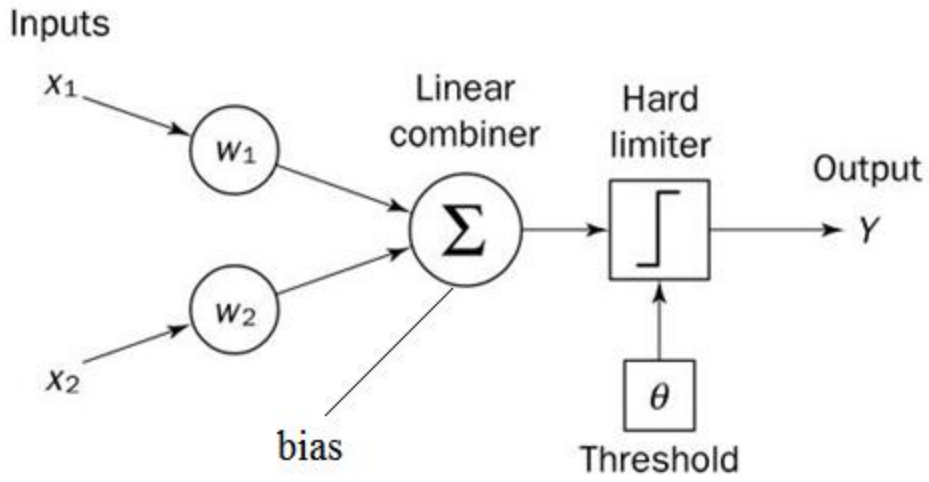


Figure 1.9: A single layer two input perceptron

In other words, weighted inputs are summed and the response so generated is then passed through a non- linear transfer function *f (*threshold or step function). The output is expressed as:

$$X = f(\sum_{i=1}^{n} w_i x_i + b\ ) \tag{1.7}$$

Correspondingly

$$Y = \begin{cases} 1\ if\ X \geq \theta \\ 0\ if\ X < \theta \end{cases} \tag{1.8}$$

where X is the net weighted input to the neuron, x_i is the value of input i, w_i is the weight of input i, n is the number of neuron inputs, and Y is the output of the neuron.

A distinct feature of Perceptron is that weights are not precalculated as in McCulloch-Pitts model of neuron but are adjusted by an iterative process called learning. A Perceptron is thus a McCulloch-Pitts model of neuron but with adjustable weights. The adaptation of weights was done using an error correction rule developed by Rosenblatt. The Perceptron could even learn when initialized with the random weight values and biases, and will converge to the correct network

weights, provided that the weights exist that could solve the problem. But the Perceptron can only classify those patterns that are linearly separable.

Perceptron is viewed as a quick and reliable network for solving a set of problems that it is designed to solve. Gaining a better understanding of the operation of perceptron enables a good understanding of more complex networks.

7. The Perceptron Learning Rule

The McCulloch-Pitts (1943) neuron model has severe limitations e.g. it lacks the capability of learning and this is attributed to the presence of fixed set of weights and threshold. To overcome these severe shortcomings, the concept of Perceptron was introduced. The Perceptron learning rule is perhaps the first of all supervised learning rules. It was introduced by Frank Rosenblatt in late 1950's. Although it is very simple in its computing capabilities, its advent influenced extensive research in the field of computing. In perceptron's, weight adaptation occurs by changing the network parameters by an amount proportional to the difference between the target output and the actual output. The learning process generally begins with a random set of weights which are then modified if the Perceptron misclassifies the given example. The Perceptron learning rule doesn't modify the weights if the error between desired and actual output is zero. This process is repeated, iterating through all the training examples as many times as needed until the Perceptron classifies all training examples correctly i.e. the training will continue until there is no error. The Perceptron learning rule converges to the correct set of weights provided the weights exist that allow the network to give correct response for the given training examples. Moreover, the net will find these weights in a finite set of iterations.

The Perceptron Learning rule involves the following basic steps:

Step 1 Initialisation

Set the initial weight values $w_1, w_2 \ldots . w_n$ and threshold θ to some random numbers.

Step 2: *Activation*

Activate the perceptron by applying inputs $x_1(p), x_2(p), \ldots . x_n(p)$ and desired output $Y_d(p)$. Calculate the actual output at iteration $p = 1$

$$Y(p) = step(\sum_{i=1}^{n} w_i(p)x_i(p) + b) \quad (1.9)$$

Where $step$ is the step activation function and n is the number of the perceptron inputs

Step 3: Weight Training

Update the weights of the perceptron

$$w_i(p+1) = w_i(p) + \Delta w_i(p) \quad (1.10)$$

Where $\Delta w_i(p)$ is the weight correction at iteration p.

The weight correction is computed by the delta rule:

$$\Delta w_i(p) = \alpha \times x_i(p) \times e(p)) \quad (1.11)$$

Where α is the learning rate parameter limited to the range $0 < \alpha \leq 1$,

$$e(p) = (Y_d(p) - Y(p)) \qquad where\ p = 1,2,3 \ldots..$$

Iteration p here refers to the pth training example presented to the perceptron.

$Y_d(p)$ is the desired response, $Y(p)$is the actual output and $e(p)$is the error.

Step 4: Iteration

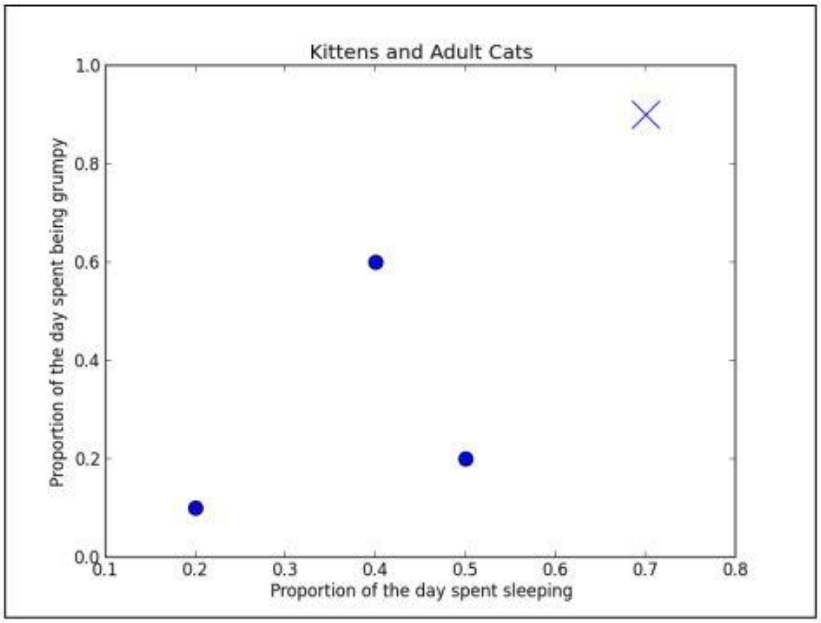
Increase iteration p by 1, go back to step2 and repeat the process until convergence.

8. Binary Classification with Perceptron

Let's work through a toy classification problem. Suppose that you wish to separate adult cats from kittens. Only two explanatory variables are available in your dataset: the proportion of the day that the animal was asleep and the proportion of the day that the animal was grumpy. Our training data consists of the following four instances:

Instance	Proportion of the day spent sleeping	Proportion of the day spent being grumpy	Kitten or Adult?
1	0.2	0.1	Kitten
2	0.4	0.6	Kitten
3	0.5	0.2	Kitten
4	0.7	0.9	Adult

The following scatter plot of the instances confirms that they are linearly separable:



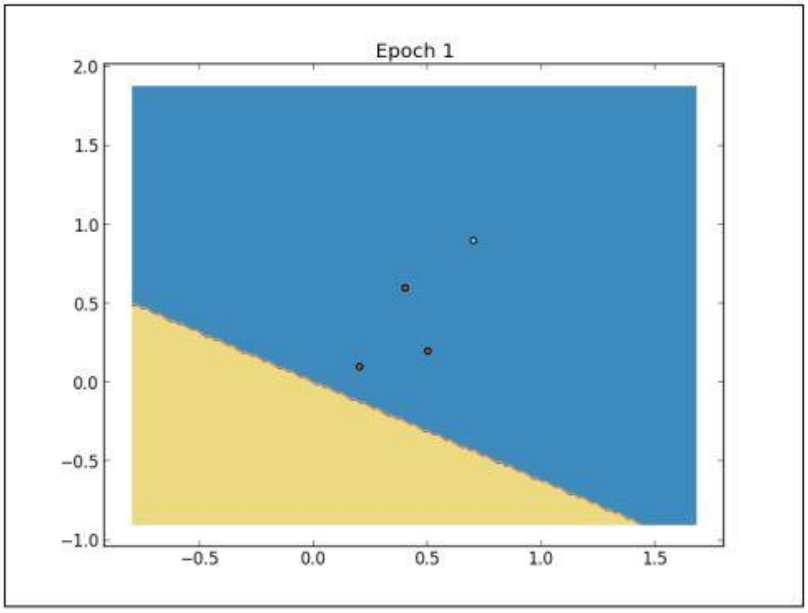
Our goal is to train a perceptron that can classify animals using the two real-valued explanatory variables. We will represent kittens with the positive class and adult cats with the negative class. The preceding network diagram describes the perceptron that we will train. Our perceptron has three input units. x_1 is the input unit for the bias term. . x_2 and . x_3 are input units for the two features.

Our perceptron's computational unit uses a Heaviside activation function. In this example, we will set the maximum number of training epochs to ten; if the algorithm does not converge within 10 epochs, it will stop and return the current values of the weights. For simplicity, we will set the learning rate to one. Initially, we will set all of the weights to zero. Let's examine the first training epoch, which is shown in the following table:

Epoch 1				
Instance	**Initial Weights x Activation**	**Prediction, Target**	**Correct**	**Updated weights**
0	0, 0, 0; 1.0, 0.2, 0.1; 1.0*0 + 0.2*0 + 0.1*0 = 0.0;	0, 1	False	1.0, 0.2, 0.1
1	1.0, 0.2, 0.1; 1.0, 0.4, 0.6; 1.0*1.0 + 0.4*0.2 + 0.6*0.1 = 1.14;	1, 1	True	1.0, 0.2, 0.1
2	1.0, 0.2, 0.1; 1.0, 0.5, 0.2; 1.0*1.0 + 0.5*0.2 + 0.2*0.1 = 1.12;	1, 1	True	1.0, 0.2, 0.1
3	1.0, 0.2, 0.1; 1.0, 0.7, 0.9; 1.0*1.0 + 0.7*0.2 + 0.9*0.1 = 1.23;	1, 0	False	0, -0.5, -0.8

Initially, all of the weights are equal to zero. The weighted sum of the explanatory variables for the first instance is zero, the activation function outputs zero, and the perceptron incorrectly predicts that the kitten is an adult cat. As the prediction was incorrect, we update the weights according to the update rule. We increment each of the weights by the product of the learning rate, the difference between the true and predicted labels and the value of the corresponding feature.

We then continue to the second training instance and calculate the weighted sum of its features using the updated weights. This sum equals 1.14, so the activation function outputs one. This prediction is correct, so we continue to the third training instance without updating the weights. The prediction for the third instance is also correct, so we continue to the fourth training instance. The weighted sum of the features for the fourth instance is 1.23. The activation function outputs one, incorrectly predicting that this adult cat is a kitten. Since this prediction is incorrect, we increment each weight by the product of the learning rate, the difference between the true and predicted labels, and its corresponding feature. We completed the first epoch by classifying all of the instances in the training set. The perceptron did not converge; it classified half of the training instances incorrectly. The following figure depicts the decision boundary after the first epoch:

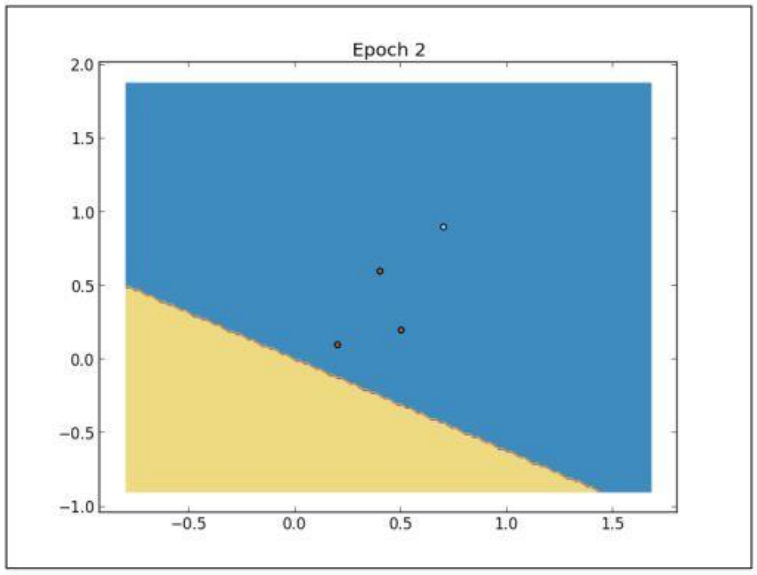


Note that the decision boundary moved throughout the epoch; the decision boundary formed by the weights at the end of the epoch would not necessarily have produced the same predictions seen earlier in the epoch. Since we have not exceeded the maximum number of training epochs, we will iterate through the instances again. The second training epoch is shown in the following table:

Epoch 2				
Instance	**Initial Weights** **x** **Activation**	**Prediction, Target**	**Correct**	**Updated weights**
0	0, -0.5, -0.8 1.0, 0.2, 0.1 1.0*0 + 0.2*-0.5 + 0.1*-0.8 = -0.18	0, 1	False	1, -0.3, -0.7
1	1, -0.3, -0.7 1.0, 0.4, 0.6 1.0*1.0 + 0.4*-0.3 + 0.6*-0.7 = 0.46	1, 1	True	1, -0.3, -0.7
2	1, -0.3, -0.7 1.0, 0.5, 0.2 1.0*1.0 + 0.5*-0.3 + 0.2*-0.7 = 0.71	1, 1	True	1, -0.3, -0.7
3	1, -0.3, -0.7 1.0, 0.7, 0.9 1.0*1.0 + 0.7*-0.3 + 0.9*-0.7 = 0.16	1, 0	False	0, -1, -1.6

The second epoch begins using the values of the weights from the first epoch. Two training instances are classified incorrectly during this epoch. The weights are updated twice, but the

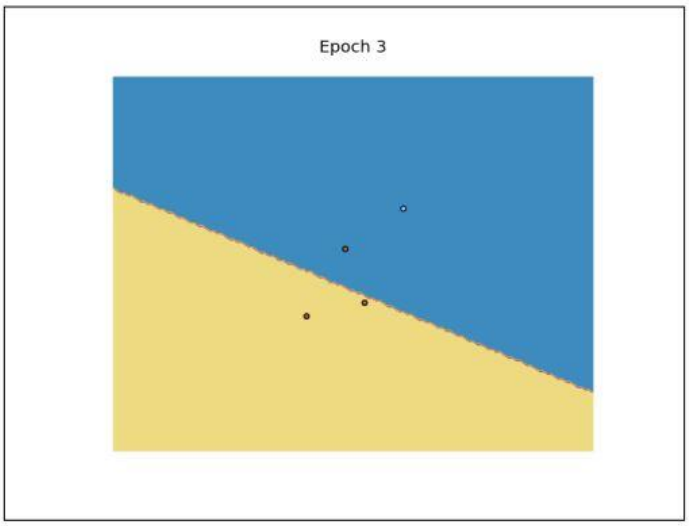
decision boundary at the end of the second epoch is similar the decision boundary at the end of the first epoch.



The algorithm failed to converge during this epoch, so we will continue training. The following table describes the third training epoch:

Epoch 3				
Instance	**Initial Weights** **x** **Activation**	**Prediction, Target**	**Correct**	**Updated Weights**
0	0, -1, -1.6 1.0, 0.2, 0.1 1.0*0 + 0.2*-1.0 + 0.1*-1.6 = -0.36	0, 1	`False`	1,-0.8, -1.5
1	1,-0.8, -1.5 1.0, 0.4, 0.6 1.0*1.0 + 0.4*-0.8 + 0.6*-1.5 = -0.22	0, 1	False	2, -0.4, -0.9
2	2, -0.4, -0.9 1.0, 0.5, 0.2 1.0*2.0 + 0.5*-0.4 + 0.2*-0.9 = 1.62	1, 1	True	2, -0.4, -0.9
3	2, -0.4, -0.9 1.0, 0.7, 0.9 1.0*2.0 + 0.7*-0.4 + 0.9*-0.9 = 0.91	1, 0	False	1, -1.1, -1.8

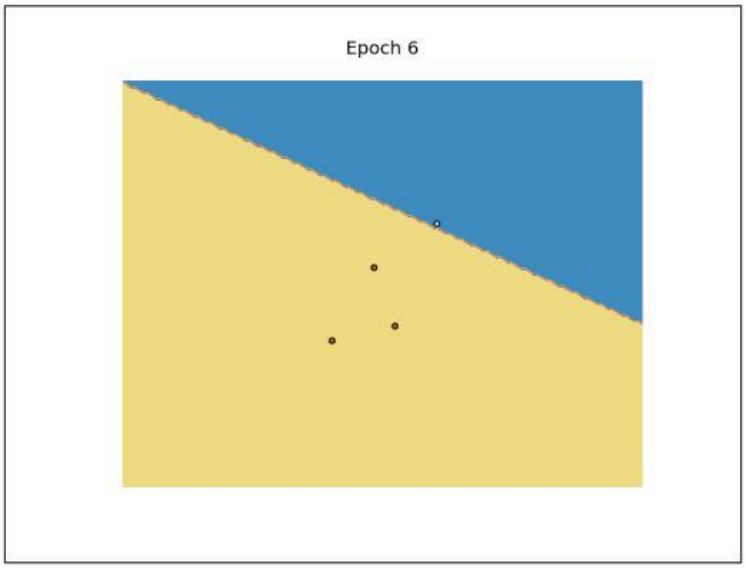
The perceptron classified more instances incorrectly during this epoch than during previous epochs. The following figure depicts the decision boundary at the end of the third epoch:



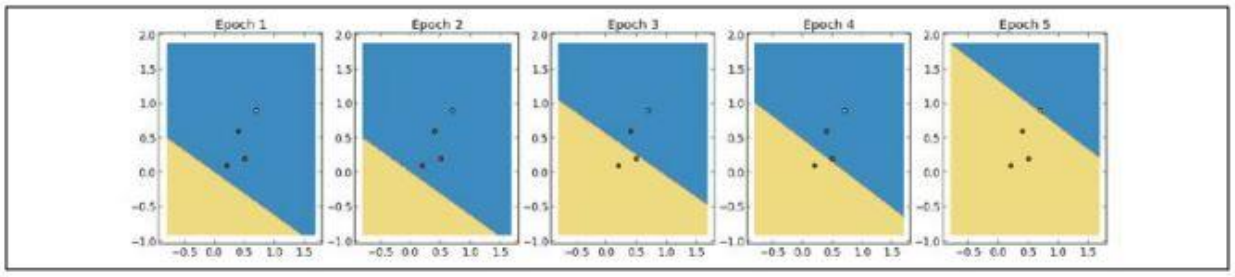
The perceptron continues to update its weights throughout the fourth and fifth training epochs, and it continues to classify training instances incorrectly. During the sixth epoch the perceptron classified all of the instances correctly; it converged on a set of weights that separates the two classes. The following table describes the sixth training epoch:

Epoch 6				
Instance	**Initial Weights** **x** **Activation**	**Prediction, Target**	**Correct**	**Updated weights**
0	2, -1, -1.5 1.0, 0.2, 0.1 1.0*2 + 0.2*-1 + 0.1*-1.5 = 1.65	1, 1	True	2, -1, -1.5
1	2, -1, -1.5 1.0, 0.4, 0.6 1.0*2 + 0.4*-1 + 0.6*-1.5 = 0.70	1, 1	True	2, -1, -1.5
2	2, -1, -1.5 1.0, 0.5, 0.2 1.0*2 + 0.5*-1 + 0.2*-1.5 = 1.2	1, 1	True	2, -1, -1.5
3	2, -1, -1.5 1.0, 0.7, 0.9 1.0*2 + 0.7*-1 + 0.9*-1.5 = -0.05	0, 0	True	2, -1, -1.5

The decision boundary at the end of the sixth training epoch is shown in the following figure:



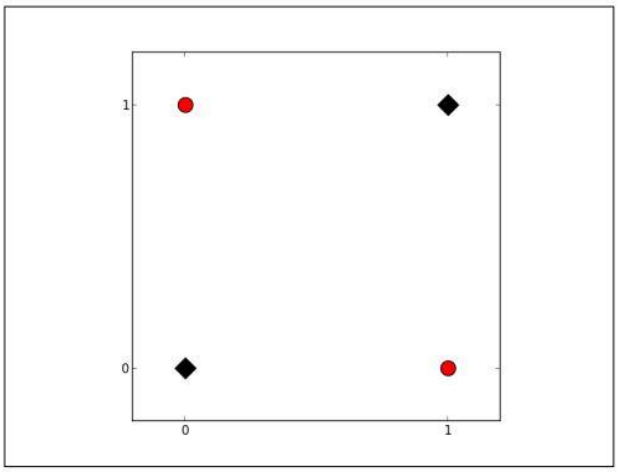
The following figure shows the decision boundary throughout all the training epochs.



9. Limitations of the perceptron

While the perceptron classified the instances in our example well, the model has limitations. Linear models like the perceptron with a Heaviside activation function are not universal function approximators; they cannot represent some functions. Specifically, linear models can only learn to approximate the functions for linearly separable datasets. The linear classifiers that we have examined find a hyperplane that separates the positive classes from the negative classes; if no hyperplane exists that can separate the classes, the problem is not linearly separable.

A simple example of a function that is linearly inseparable is the logical operation XOR, or exclusive disjunction. The output of XOR is one when one of its inputs is equal to one and the other is equal to zero. The inputs and outputs of XOR are plotted in two dimensions in the following graph. When XOR outputs 1, the instance is marked with a circle; when XOR outputs 0, the instance is marked with a diamond, as shown in the following figure:



It is impossible to separate the circles from the diamonds using a single straight line.

10. Multilayer Neural Networks

A multilayer perceptron is a feedforward neural network with one or more hidden layers. Typically, the network consists of an input layer of source neurons, at least one middle or hidden layer of computational neurons, and an output layer of computational neurons. The input signals are propagated in a forward direction on a layer-by-layer basis. A multilayer perceptron with two hidden layers is shown in Figure 2.

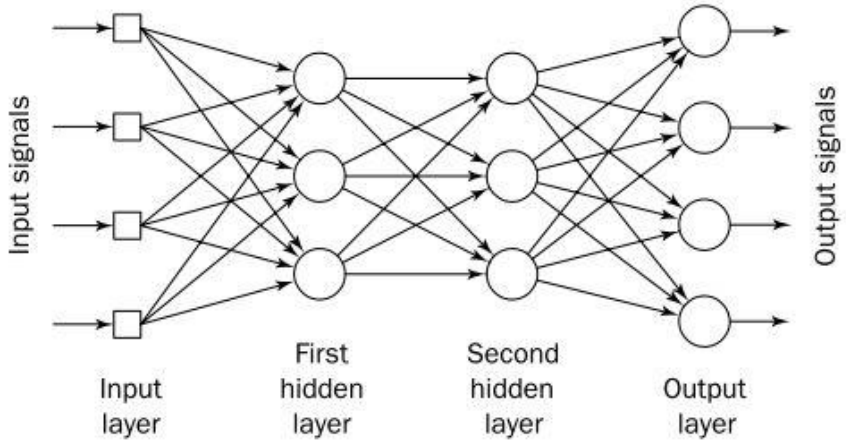


Figure 2. A multilayer perceptron

A hidden layer ‘hides’ its desired output. Neurons in the hidden layer cannot be observed through the input/output behaviour of the network. There is no obvious way to know what the desired output of the hidden layer should be. In other words, the desired output of the hidden layer is

determined by the layer itself. The purpose of the hidden units is to introduce non linearity in the approximated relationship between inputs and the outputs and the number of hidden neurons determines the level of non linearity that the NN is capable of representing. Hence if number of hidden units used is less, it will be impossible to map the training data precisely because the information capacity of the neural network will not correspond to the amount of the information represented by the training data. On the other hand if large number of hidden units are used, the information capacity of the neural network will exceed the necessary minimum and the redundant hidden units will be able to learn the information such as the order of patterns or data noise, thus reducing the ability of neural network to correctly predict the output of the previously unseen patterns. It is thus necessary to have correct number of hidden units.

10.1 Learning in Multilayer Perceptron

The MLP is one of the most popular and most frequently used types of ANN models due to its clear architecture and comparably simple algorithm. The MLP is composed of a set of sensorial nodes organised in a hierarchy of layers, which comprise of the input layer of nodes, one or more intermediary or hidden(s) layer of computational nodes, and the output layer of nodes that calculates the output of the network. Typically MLP's are trained with Back propagation (BP) algorithm.

Backpropgation is a renowned supervised form of learning algorithm, developed by [Rumelhart & McClelland, 1986]. It was developed in order to overcome the drawback of previous ANN algorithm where single layer perceptron fail to solve a simple XOR problem. The Perceptron can only be used for the classes that are linearly separable whereas a typical multilayer network, which is learned by the BP algorithm, has the aptitude to classify the classes which are not necessarily linearly separable This type of ANN algorithm is a supervised learning algorithm since it requires a desired output in order to learn the network. The BP algorithm is basically generalization of the LMS (Least-Mean-Squares) algorithm. The idea of the BP is to increase training speed, improve performance and obtain desired output values by adjusting weights and scaling inputs. Selecting the neural network structure is extremely important in classification problems. This may also cause overfitting or underfitting for the training samples.

BP is based on the GD method that endeavours to minimise the error of the network by moving down the gradient of error curve. In BP the error at each output units is propagated back to yield estimates of how much a given hidden unit contributed to the output error. These estimates are used in the adjustment of synaptic weights to these units within the network. In its simplest form training begins by presenting a training set of input patterns and the corresponding desired target patterns for the output units. The activations of the input nodes are multiplied by the weighted connections and are passed through a transfer function at each node in the first hidden layer. The activations from the first hidden layer are then passed to the neurons in the next layer, and this process is repeated until the output activations are obtained from the output layer. The output activation values and the target pattern are compared and the error signal is calculated based on the difference between target and calculated pattern. This error signal is then propagated backwards to adjust network weights so that network will generate correct output for the presented input pattern. The training patterns are presented repeatedly until the error reaches an acceptable value or other convergence criteria are satisfied. As this technique involves performing computation backwards it is named as Back Propagation.

BP can be partitioned into two stages, a forward pass and backward pass. During the forward pass, this algorithm maps the input values to the output through the network. All input vectors are presented to the input layer, the outputs of input neurons corresponds to component vectors. Net activation is computed in hidden neurons which take weighted sum of inputs into account. Briefly, net activation is the sum of the inner products of weight and input vectors added by bias. The generated output pattern is obtained from a summation of the weighted input of node and by applying the network activation function (sigmoid function). In the backward pass, this output pattern (actual output) is then compared to the desired output and the error signal is computed for each output unit. The error signals are then transmitted backward from the output layer to each unit in the transitional layer that contributes directly to the output and the weights are adjusted iteratively during the learning process, thus the error is reduced along a descent direction.

11.Backpropagation Learning Algorithm

To derive the back-propagation learning law, let us consider the three-layer network shown in Figure 6.9. The indices i, j and k here refer to neurons in the input, hidden and output layers, respectively. Input signals, $x_1, x_2 \ldots . x_n$. are propagated through the network from left to right,

and error signals, $e_1, e_2 \ldots . e.$ from right to left. The symbol w_{ij} denotes the weight for the connection between neuron i in the input layer and neuron j in the hidden layer, and the symbol w_{jk} the weight between neuron j in the hidden layer and neuron k in the output layer.

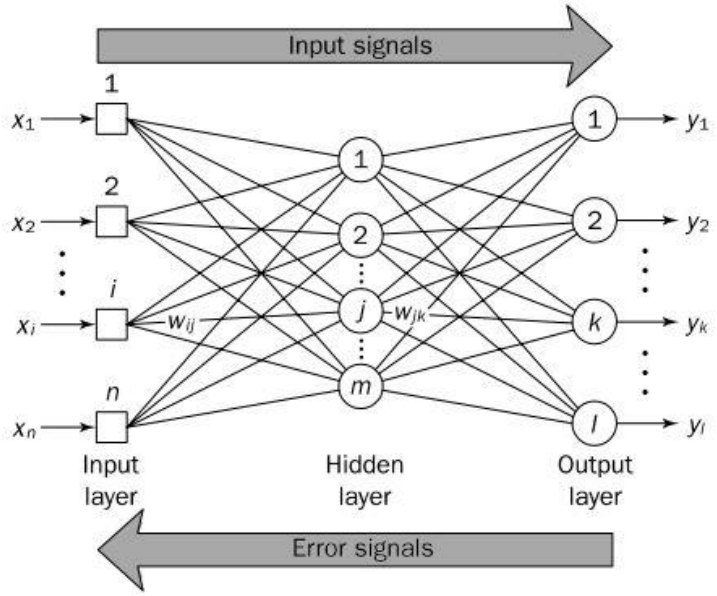


Figure 2.1 Three layer Backpropagation Neural Network

To propagate error signals, we start at the output layer and work backward to the hidden layer. The error signal at the output of neuron k at iteration p is defined by

$$e_k(p) = (y_{d,k}(p) - y(p)) \tag{1.12}$$

where $y_{d,k}(p)$ is the desired output of neuron k at iteration p.

Neuron k, which is located in the output layer, is supplied with a desired output of its own. Hence, we may use a straightforward procedure to update weight w_{jk}. In fact, the rule for updating weights at the output layer is similar to the perceptron learning rule of Eq. (1.10):

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p) \tag{1.13}$$

where $\Delta w_{jk}(p)$ is the weight correction term.

Weight Correction term for neuron in output layer

When we determined the weight correction for the perceptron, we used input signal x_i. But in the multilayer network, the inputs of neurons in the output layer are different from the inputs of neurons in the input layer. However, we use the output of neuron$j$ in the hidden layer, y_j, instead of input x_i. The weight correction in the multilayer network is computed by

$$\Delta w_{jk}(p) = \propto \times y_j(p) \times \delta_k(p) \tag{1.14}$$

where $\delta_k(p)$ is the error gradient at neuron k in the output layer at iteration p.

The error gradient is determined as the derivative of the activation function multiplied by the error at the neuron output.

$$\delta_k(p) = y_k(p)\ (1 - y_k(p))\ (e_k(p)) \tag{1.15}$$

where

$$(y_k(p)) = {}^{1}/_{1 + e^{-(X_k(p))}}$$

Weight Correction term for neuron in hidden layer

To calculate the weight correction for the hidden layer, we can apply the same equation as for the output layer:

$$\Delta w_{ij}(p) = \propto \times x_i(p) \times \delta_j(p) \tag{1.16}$$

where $\delta_j(p)$ represents the error gradient at neuron j in the hidden layer:

$$\delta_j(p) = y_j(p)(\ 1 - y_j(p))\ (\sum_{k=1}^{l} w_{jk}(p)\ \ \delta_k(p))$$

where l is the number of neurons in the output layer;

$$y_j(p) = {}^{1}/_{1 + e^{-(X_j(p))}};$$

$$X_j(p) = \sum_{i=1}^{n} w_{ij}(p) x_i(p) + b_j)$$

and n is the number of neurons in the input layer.

Hence from the preceding discussion the BP algorithm can be summarized as follows:

Step 1:Initialization

Initialize the values of w_{ij}, w_{jk} to small random values within the range [0, 1].

Step 2:Activation

Activate the back-propagation neural network by applying inputs

$x_1(p), x_2(p) \ldots . x_n(p)$ and desired ouputs $y_{d,1}(p),\ y_{d,2}(p) \ldots .\ y_{d,n}(p)(p)$

a. Calculate the actual outputs of the neurons in the hidden layer:

$$y_j(p) = sigmoid(\sum_{i=1}^{n} w_{ij}(p)x_i(p) + b_j)$$

where n is the number of inputs of neuron$j$ in the hidden layer, and *sigmoid* is the sigmoid activation function.

b. Calculate the actual outputs of the neurons in the output layer:

$$y_k(p) = sigmoid(\sum_{j=1}^{m} w_{jk}(p)x_{jk}(p) + b_k)$$

where m is the number of inputs of neuron k in the output layer.

Step 3: Weight training

Update the weights in the back-propagation network propagating backward the errors associated with output neurons.

a. Calculate the error gradient for the neurons in the output layer:

$$\delta_k(p) = y_k(p)\ (1 - y_k(p))\ (e_k(p))$$

Where

$$e_k(p) = (y_{d,k}(p) - y(p))$$

Calculate the weight corrections:

$$\Delta w_{jk}(p) = \propto \times y_j(p) \times \delta_k(p)$$

Calculate the bias corrections:

$$\Delta b_k(p) = \propto \times \delta_k(p)$$

Update the weights and bias at the output neurons:

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p)$$

$$b_k(p+1) = b_k(p) + \Delta b_k(p)$$

b. Calculate the error gradient for the neurons in the hidden layer:

$$\delta_j(p) = y_j(p)(1 - y_j(p)) \left(\sum_{k=1}^{l} w_{jk}(p)\ \delta_k(p)\right)$$

Calculate the weight corrections:

$$\Delta w_{ij}(p) = \propto \times x_i(p) \times \delta_j(p)$$

Calculate the bias corrections:

$$\Delta b_j(p) = \propto \times \delta_j(p)$$

Update the weights and biases at the hidden neurons:

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p)$$

$$b_j(p+1) = b_j(p) + \Delta b_j(p)$$

Step 4: Iteration

Increase iteration p by one, go back to Step 2 and repeat the process until the selected error criterion is satisfied.

11.1 Accelerated learning in Multilayer Neural Networks

The computational efficiency of BP algorithm can be accelerated by including a momentum term in the Equation 1.14; the weight update rule after the introduction of momentum term is given as:

$$\Delta w_{jk}(p) = \beta \times \Delta w_{jk}(p-1) + \alpha \times y_j(p) \times \delta_k(p)$$

where β is a positive number $(0 \leq \beta < 1)$ is called the momentum constant.

Typically, the momentum constant is set to 0.95. if the momentum is added to the weight update formula, the convergence is faster. The weights from one or more previous training patterns must be saved in order to use the momentum. For the BPNN with momentum, the net weights for training step $(p+2)$, is based on p and $p+1$. It is found that momentum allows net to perform large weight adjustments as long as the correction proceeds in the same general direction for several patterns. Thus using momentum, the net does not proceed in the direction of gradient, but travels in the direction of the combination of the current gradient and the previous direction for which the weight correction is made. The main purpose of the momentum is to accelerate the convergence of error propagation algorithm. This method makes the current weight adjustment with the fraction of recent weight adjustment.

11.2 Training style

Updation of weights in BP can be done in either of the two ways:

(i) Online updating or Pattern by Pattern updating, where the weights are updated after every pattern presentation. This type of learning is recommended for applications requiring high accuracy and can compromise on other factors such as time etc.

(ii) Batch or Epoch Based Training, where weight adjustments are made only at the end of epoch i.e. after the presentation of the entire training set. In both the schemes the learning process continues until the error reaches a predefined value.

Find the new weights when the network illustrated in Fig. 2.4 is presented the input pattern[0.6,0.8] and target output is 0.9. use learning rate 0.3 and use sigmoid function.

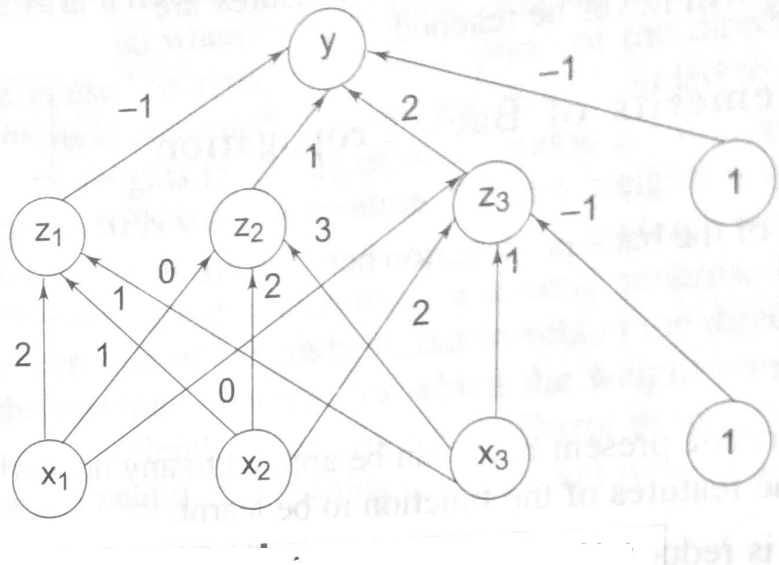


Figure 2.4: Backpropagation Net

Solution The solution to the problem is as follows:

Step 1: Initialize the weight and bias

$$W = [-1\ 1\ 2],\ W_0 = [-1]$$

$$v = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 2 \\ 0 & 3 & 1 \end{bmatrix}\ V_0 = [0\ \ 0\ \ -1]$$

Step 3: For each training pair

$$X = [0.6\ \ 0.8\ \ 0]$$

$$t = [0.9]$$

Feed Forward Stage

Step 4:

$$z_{-inj} = v_{oj} + \sum_{i=1}^{n} x_i v_{ij}$$

$$\Rightarrow Z_{-in1} = V_{o1} + \sum_{i=1}^{3} x_i v_{i1}$$

$$\Rightarrow V_{o1} + x_1 V_{11} + X_2 V_{21} + X_3 V_{31}$$

$$\Rightarrow 0 + 0.6 \times 2 + 0.8 \times 1 + 0 \times 0$$

$$\Rightarrow 1.2 + 0.8 = 2$$

$$z_{-in2} = v_{o2} + \sum_{i=1}^{3} x_i v_{i2}$$

$$\Rightarrow V_{o2} + x_1 V_{12} + X_2 V_{22} + X_3 V_{32}$$

$$\Rightarrow 0 + 0.6 + 0.8 \times 2 + 0 \times 3$$

$$\Rightarrow 2.2$$

$$Z_{-in3} = v_{o3} + \sum_{i=1}^{3} x_i v_{i3}$$

$$\Rightarrow V_{03} + X_1 V_{13} + X_2 V_{23} + X_3 V_{33}$$

$$\Rightarrow -1 + 0.6 \times 0 + 0.8 \times 2 + 0 \times 1$$

$$\Rightarrow -1 + 1.6 = 0.6$$

$$z_1 = f(z_{-in1}) = \frac{1}{1 + e^{-2}} = 0.8808$$

$$z_2 = f(z_{-in2}) = \frac{1}{1 + e^{-2.2}} = 0.9002$$

$$z_3 = f(z_{-in3}) = \frac{1}{1 + e^{-0.6}} = 0.646$$

Step 5: Calculate Y_{-ink}

$$y_{-ink} = w_{ok} + \sum_{j=1}^{p} z_j w_{jk}$$

$$y_{-in1} = w_{o1} + \sum_{j=1}^{3} z_j w_{j1}$$

$$= w_{01} + Z_1 W_{11} + Z_2 W_{21} + Z_3 W_{31}$$

$$\Rightarrow -1 + 0.8808 \times -1 + 0.9002 \times 1 + 0.646 \times 2$$

$$\Rightarrow -1 - 0.8808 + 0.9002 + 1.292$$

$$y_{-in1} = [0.3114]$$

Calculate the output signal

$$y_1 = f[y_{-in1}] = \frac{1}{1 + e^{-0.3114}} = 0.5772$$

Back Propagation of Error

Step 6: Calculate error information term δ_k.

$$\delta_k = (t_k - y_k)\, f^1(y_{-ink})$$

$$\delta_1 = (t_1 - y_1)\, f^1\,(y_{-in1})$$

we know that for binary sigmoid function

$$f^1(x) = f(x)(1 - f(x))$$

$$f^1(y_{-in1}) = f(y_{-in1})(1 - f(y_{-in1}))$$

$$= 0.5772\,(1 - 0.5772)$$

$$= 0.2440$$

$$\delta_1 = (t_1 - y_1)\, f^1(y_{-in1})$$

$$\Rightarrow (0.9 - 0.5772)(0.2440)$$

$$= 0.0708$$

Step 7: Back propagation to be first hidden layer ($i = 1,2,3$) we have

To calculate δ_{-in1}

$$\delta_{-inj} = \sum_{k=1}^{m} \delta_k w_{jk}$$

$$\delta_{-inj} = \sum_{k=1} \delta_k w_{jk}$$

$$\delta_{-in1} \Rightarrow \delta_1 w_{11} \Rightarrow 0.0788 \times -1 = -0.0788$$

$$\delta_{-in2} = \delta_1 w_{21} = 0.0788 \times 1 = 0.0788$$

$$\delta_{-in3} = \delta_1 w_{31} \Rightarrow 0.0788 \times 2 = 0.01576$$

To calculate error term in hidden layer,

$$\Delta = \delta_{-inj}\, f^1(z_{-inj})$$

$$f^1(z_{-in1}) = f(z_{-in1})\,(1 - f_{-in1})$$

$$\Rightarrow (0.8808)(1 - 0.8808)$$

$$= 0.1049$$

$$\Delta_1 = \delta_{-in1}\, f(z_{-in1})\,(1 - f_{-zin1})$$

$$\Rightarrow (-0.0788)(0.1049)$$

$$= -0.0083$$

$$\Delta_2 = \delta_{-in2}\, f^1(z_{-in2})$$

$$f^1(z_{-in2}) = f(z_{-in2})(1 - f_{zin2})$$

$$\Rightarrow (0.9002)\,(1 - 0.9002)$$

$$= 0.09$$

$$\Delta_2 = 0.0788 \times 0.09 \Rightarrow 0.0071$$

$$\Delta_3 = \delta_{-in3}\, f^1(z_{-in3})$$

$$f^1(z_{-in3}) = f(z_{-in3})(1 - f(z_{-in3}))$$

$$\Rightarrow (0.646)(1 - 0.646)$$

$$\Rightarrow 0.2286$$

$$\Delta_3 = 0.1576 \times 0.2286$$

$$= 0.0361$$

Step 8: Weight Updation

$$\Delta v_{ij} = \alpha \Delta_j x_i$$

$$x = [0.6\ 0.8\ 0]$$

$$\Delta = [-0.0083\ 0.0071\ 0.0361]$$

$$\alpha = 0.3$$

$$\Delta v_{11} = \alpha \Delta_1 x_1 = 0.3 \times -0.0083 \times 0.6 = -0.0015$$

$$\Delta v_{12} = \alpha \Delta_2 x_i = 0.3 \times 0.0071 \times 0.6 = 0.0013$$

$$\Delta v_{21} = \alpha\Delta_1 x_2 = 0.3 \times -0.0083 \times 0.8 = -0.002$$

$$\Delta v_{22} = \alpha\Delta_2 x_2 = 0.3 \times 0.0071 \times 0.8 = 0.0017$$

$$\Delta v_{13} = \alpha\Delta_3 x_1 = 0.3 \times 0.0361 \times 0.6 = 0.0065$$

$$\Delta v_{23} = \alpha\Delta_3 x_2 = 0.0087$$

$$\Delta v_{31} = \Delta_{32} = \Delta v_{33} = 0$$

$$\Delta v_{01} = \alpha\Delta_1;\ \Delta v_{02} = \alpha\Delta_2;\qquad \Delta v_{03} = \alpha\Delta_3$$

$$\Delta v_0 = 0.3 \times -0.0023 \Rightarrow -0.0025$$

$$\Delta v_{02} = 0.3 \times 0.0071 \Rightarrow 0.0021$$

$$\Delta v_{03} = 0.3 \times 0.0361 \Rightarrow 0.0108$$

$$v_{new} = v_{old} + \Delta v_1$$

$$v_{11}(new) = v_{11}(old) + \Delta v_{11} = 2 - 0.0015 = 1.9985$$

$$v_{12}(new) = v_{12}(old) + \Delta v_{13} = 9 + 0.0013 = 1.0013$$

$$v_{13}(new) = v_{13}(old) + \Delta v_{13} = 01 + 0.065 = 0.065$$

$$v_{21}(new) = v_{21}(old) + \Delta v_{21} = 1 - 0.002 = 0.998$$

$$v_{22}(new) = v_{22}(old) + \Delta v_{22} = 2 + 0.0017 = 2.0017$$

$$v_{23}(new) = v_{23}(old) + \Delta_{23} = 2 + 0.0067 = 2.0087$$

$$v_{31}(new) = v_{31}(old) + \Delta_{31} = 0 + 0 = 0$$

$$v_{32}(new) = v_{32}(old) + \Delta_{32} = 3 + 0 = 3$$

$$v_{33}(new) = v_{32}(old) + \Delta_{33} = 1 + 0 = 1$$

$$\therefore v = \begin{bmatrix} 1.9985 & 1.0013 & 0.085 \\ 0.998 & 2.0017 & 2.0087 \\ 0 & 3 & 1 \end{bmatrix}$$

$$\Delta W_{ok} = \alpha\delta_k z_j$$

$$\Delta W_{11} = \alpha\delta_1 z_1 = 0.3 \times 0.0788 \times 0.8808 = 0.0208$$

$$\Delta W_{12} = \propto\delta_1 z_2 = 0.3 \times 0.0788 \times 0.9002 = 0.0212$$

$$\Delta W_{13} = \propto\delta_1 z_3 = 0.3 \times 0.0788 \times 0.646 = 0.0153$$

$$\Delta W_{11}(new) = w_{11}(old) + \Delta w_{11} = -1 + 0.0208 = 0.9792$$

$$\Delta W_{12}(new) = w_{12}(old) + \Delta w_{12} = 1 + 0.0212 = 1.0212$$

$$\Delta W_{13}(new) = w_{13}(old) + \Delta w_{13} = 2 + 0.0153 = 2.0153$$

$$\Delta W_0 = \alpha s_1 = 0.3 \times 0.0788 = 0.02364$$

$$w = [0.9792 \quad 1.0212 \quad 2.0153]$$

Thus the weights are calculated. The process can be continued upto any specified stopping criteria

12.Synergistic Neural Networks

It has been shown for different applications that combining a number of unit structure neural networks can yield better results than achievable by a single unit structure model(one neural network unit structure for modelling a data set) on its own. Combined neural networks are called 'synergistic' networks (Fig. 5), the term synergistic being derived from Greek words meaning 'joint efforts'. The essence of the synergistic approach is to employ m unit structures, each slightly different from others so that they can yield slightly different outputs to a given set of inputs. A combining module then uses the outputs of the individual unit structures and selects the best output as the final result. The m unit structures are made different by using different neuron characteristics (e.g. different activation functions) in the unit structures. The use of different activation functions appears to be a good choice, as different activation functions may suit different non-linearities that may be present in a data set.

The principle of operation of this synergistic system can be likened to that of a group of experts, each with different backgrounds and experiences. This makes them produce different solutions to a given problem. Another expert then examines these solutions and synthesizes the final group solution. To achieve synergy, it is important that the individual neural models are different from one another, just as human experts are. Various ways exist to ensure this, including :

(a) using different types of neural models [for example mixing multi-layer perceptrons and learning vector quantization networks ;

(b) training the neural models using different data sets;

(c) using neural models with different internal configurations (different numbers of hidden neurons, for instance);

(d) using neural models with different neuron characteristics (different activation functions);

(e) teaching the neural models with different starting conditions, training parameters and/or learning algorithms.

Adopting the synergistic approach increases the accuracy of the overall model, making it higher than that of the best individual neural model in a given combination. It is interesting to note that

increasing the number of neural models does not necessarily improve the overall performance. This could be explained by the fact that, when a large number of neural models is employed, the resulting combination process is more complex and therefore it is more difficult to train the neural combiner well. A possible disadvantage with the synergistic approach is the need to train more than one neural network-at least three (two neural models and one neural combiner). However, as the individual neural networks all possess very simple structures, they are easy to train.

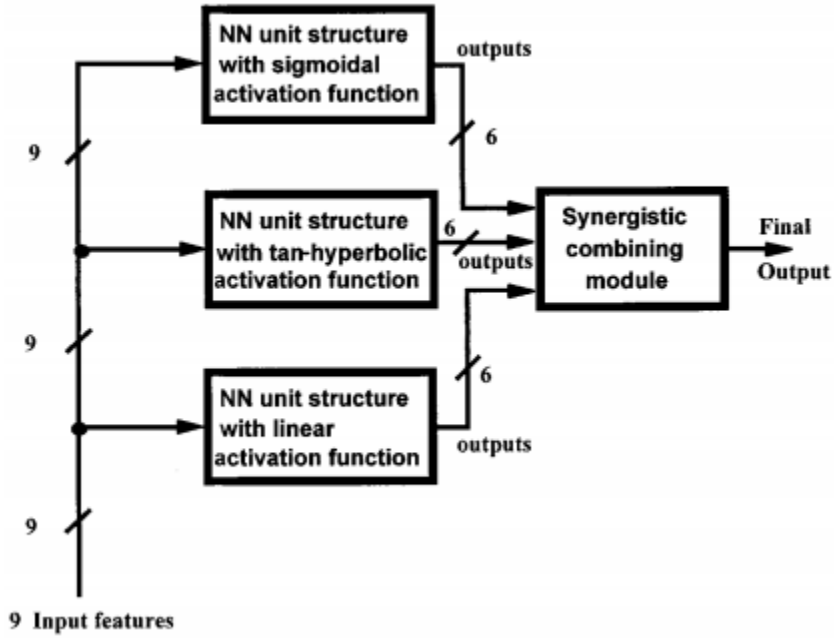


Figure 2.2 Synergistic Multistructure Neural Network

13.Distributed neural network.

Another approach for improving the recognition accuracy is to use a distributed multistructure neural network. A distributed multistructure neural network has many interconnected unit structure neural networks where each unit structure learns a part of the characteristics of a given data set. Thus the characteristics of a data set are trained in parts on various unit structures instead of training the entire characteristics of a data set on a single unit structure. An example of a distributed multistructure neural network is shown in Fig. 2.3. The essence of the distributed approach is to make easy the difficult task of learning all classes on a single unit structure neural network to suit the situations where some characteristics of a data set are shared by more than one class. Such data

sets can be trained on a distributed multistructure neural network where each unit structure neural network differentiates only one class from the rest of the classes. The interconnected unit structures thus jointly differentiate all classes that are present in a given data set. This improves the accuracy of recognition in situations where overlapping ranges of values corresponding to various classes are present in training and testing data sets.

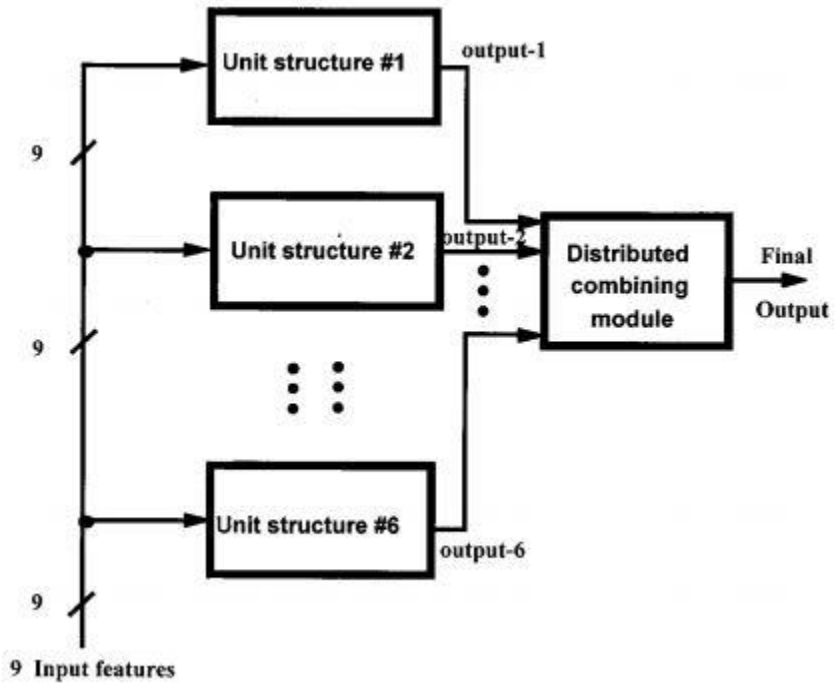


Figure 2.3 Distributed Neural Network

14.Distributed Synergistic Neural Network.

Distributed synergistic multistructure neural network. To improve further the accuracy of recognition, the features of synergistic and distributed multistructure neural networks can be combined to form a new distributed synergistic multistructure neural network. A distributed synergistic neural network has many interconnected unit structure neural networks that can cope with non-linear characteristics and overlapping ranges of values associated with a given data set. An example of a distributed synergistic multistructure neural network model is shown in Fig. 2.4. As pointed out above, unit structures with different neuron characteristics are used in a synergistic

model for better recognition accuracy as different activation functions may suit different non-linearities that may be present in a given data set. In a distributed multistructure model the characteristics of a data set are trained in parts on various unit structures for better recognition accuracy as it attempts to recognize only a part of the characteristics of the data set on one unit structure rather than recognizing the entire characteristics on a single unit structure.

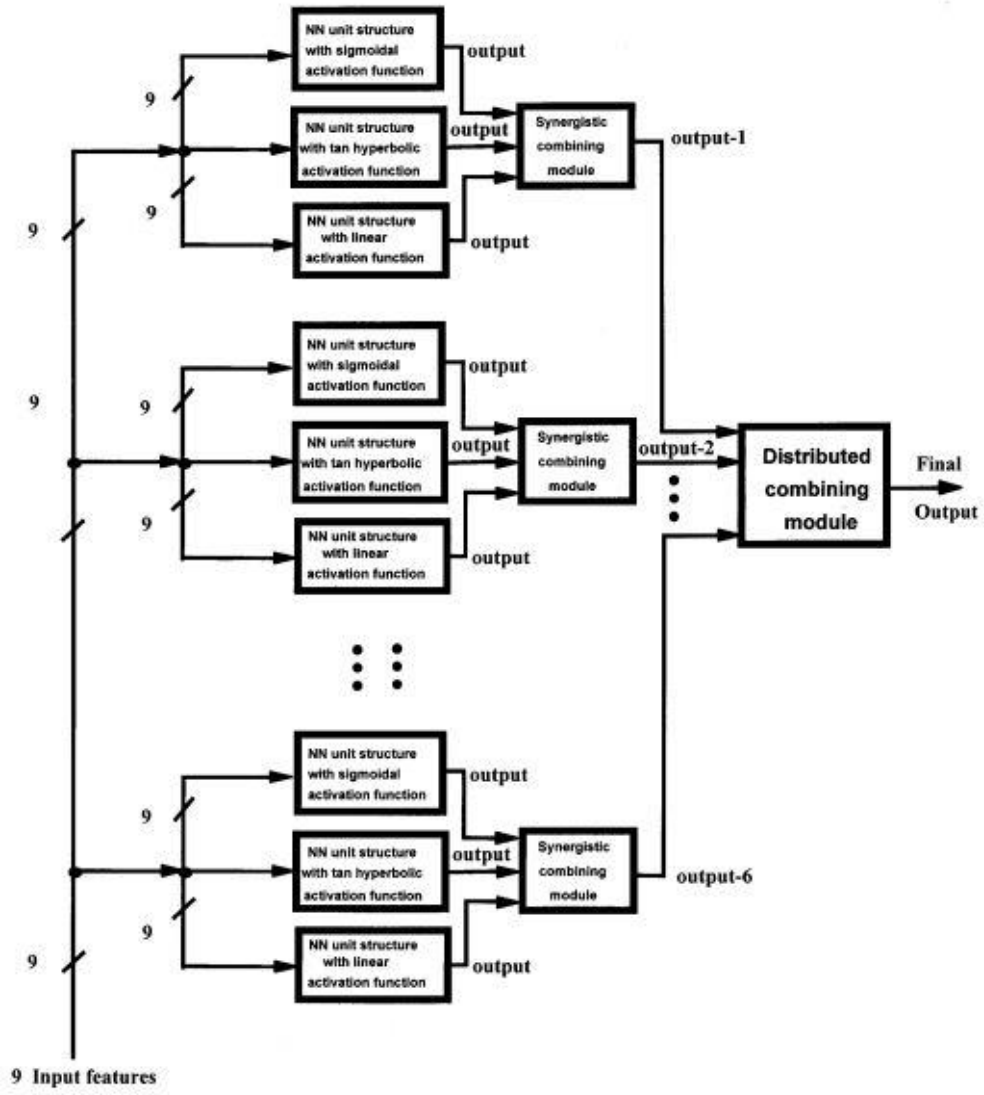


Figure 2.4: Distributed Synergistic Neural Network

15. Applications of AI

(i) Banking

AI in banking is growing faster, a lot of banks have already adopted AI-based systems to provide customer support, detect anomalies and credit card frauds. An example of this is HDFC Bank. HDFC Bank has developed an AI-based chatbot called EVA (Electronic Virtual Assistant), built by Bengaluru-based Senseforth AI Research. Since its launch, Eva has addressed over 3 million customer queries, interacted with over half a million unique users, and held over a million conversations. Eva can collect knowledge from thousands of sources and provide simple answers in less than 0.4 seconds. The use of AI for fraud prevention is not a new concept. In fact, AI solutions can be used to enhance security across a number of business sectors, including retail and finance.

By tracing card usage and endpoint access, security specialists are more effectively preventing fraud. Organizations rely on AI to trace those steps by analyzing the behaviors of transactions.

Companies such as MasterCard and RBS WorldPay have relied on AI and Deep Learning to detect fraudulent transaction patterns and prevent card fraud for years now. This has saved millions of dollars.

(ii) Finance

Ventures have been relying on computers and data scientists to determine future patterns in the market. Trading mainly depends on the ability to predict the future accurately. Machines are great at this because they can crunch a huge amount of data in a short span. Machines can also learn to observe patterns in past data and predict how these patterns might repeat in the future. In the age of ultra-high-frequency trading, financial organizations are turning to AI to improve their stock trading performance and boost profit.

(iii) Agriculture

AI can help farmers get more from the land while using resources more sustainably. Issues such as climate change, population growth, and food security concerns have pushed the industry into seeking more innovative approaches to improve crop yield. Organizations are using automation and robotics to help farmers find more efficient ways to protect their crops from weeds. AI can be

used to identify the potential defects and nutrient deficiencies in the soil by providing users with soil restoration techniques, tips, and other possible solutions.

(iv) Gaming

Over the past few years, Artificial Intelligence has become an integral part of the gaming industry. In fact, one of the biggest accomplishments of AI is in the gaming industry. DeepMind's AI-based AlphaGo software, which is known for defeating Lee Sedol, the world champion in the game of GO, is considered to be one of the most significant accomplishment in the field of AI. Shortly after the victory, DeepMind created an advanced version of AlphaGo called AlphaGo Zero which defeated the predecessor in an AI-AI face off. Unlike the original AlphaGo, which DeepMind trained over time by using a large amount of data and supervision, the advanced system, AlphaGo Zero taught itself to master the game.

Other examples of Artificial Intelligence in gaming include the First Encounter Assault Recon, popularly known as F.E.A.R, which is a first-person shooter video game.

(v) Space Exploration

Space expeditions and discoveries always require analyzing vast amounts of data. Artificial Intelligence and Machine learning is the best way to handle and process data on this scale. After rigorous research, astronomers used Artificial Intelligence to sift through years of data obtained by the Kepler telescope in order to identify a distant eight-planet solar system.

Artificial Intelligence is also being used for NASA's next rover mission to Mars, the Mars 2020 Rover. The AEGIS, which is an AI-based Mars rover is already on the red planet. The rover is responsible for autonomous targeting of cameras in order to perform investigations on Mars.

(vi) Autonomous Vehicles

For the longest time, self-driving cars have been a buzzword in the AI industry. The development of autonomous vehicles will definitely revolutionaries the transport system. Companies like Waymo conducted several test drives in Phoenix before deploying their first AI-based public ride-hailing service. The AI system collects data from the vehicles radar, cameras, GPS, and cloud services to produce control signals that operate the vehicle. Advanced Deep

Learning algorithms can accurately predict what objects in the vehicle's vicinity are likely to do. This makes Waymo cars more effective and safer.

Another famous example of an autonomous vehicle is Tesla's self-driving car. Artificial Intelligence implements computer vision, image detection and deep learning to build cars that can automatically detect objects and drive around without human intervention.

(vii) Chatbots

Virtual assistants have become a very common technology. Almost every household has a virtual assistant that controls the appliances at home. A few examples include Siri, Cortana, which are gaining popularity because of the user experience they provide.

Amazon's Echo is an example of how Artificial Intelligence can be used to translate human language into desirable actions. This device uses speech recognition and NLP to perform a wide range of tasks on your command. It can do more than just play your favorite songs. It can be used to control the devices at your house, book cabs, make phone calls, order your favorite food, check the weather conditions and so on.

Another example is the newly released Google's virtual assistant called Google Duplex, that has astonished millions of people. Not only can it respond to calls and book appointments for you, but it also adds a human touch. The device uses Natural language processing and machine learning algorithms to process human language and perform tasks such as manage your schedule, control your smart home, make a reservation and so on.

(viii) Social Media

Ever since social media has become our identity, we've been generating an immeasurable amount of data through chats, tweets, posts and so on. And wherever there is an abundance of data, AI and Machine Learning are always involved.

In social media platforms like Facebook, AI is used for face verification wherein machine learning and deep learning concepts are used to detect facial features and tag your friends. Deep Learning is used to extract every minute detail from an image by using a bunch of deep neural networks. On the other hand, Machine learning algorithms are used to design your feed based on your interests.

Another such example is Twitter's AI, which is being used to identify hate speech and terroristic language in tweets. It makes use of Machine Learning, Deep Learning, and Natural language processing to filter out offensive content. The company discovered and banned 300,000 terrorist-linked accounts, 95% of which were found by non-human, artificially intelligent machines.

(ix) Artificial Creativity

An AI based system to create music and art that echoes the classical legends. MuseNet is a deep neural network that is capable of generating 4-minute musical compositions with 10 different instruments and can combine styles MuseNet was not explicitly programmed with an understanding of music, but instead discovered patterns of harmony, rhythm, and style by learning on its own.

Another creative product of Artificial Intelligence is a content automation tool called Wordsmith. Wordsmith is a natural language generation platform that can transform your data into insightful narratives.

(x) Health care

AI has countless applications in healthcare. Whether it's being used to discover links between genetic codes, to power surgical robots or even to maximize hospital efficiency, AI has been a boon to the healthcare industry.