RGPVNOTES.IN

Please do not share these notes on apps like WhatsApp or Telegram.

The revenue we generate from the ads we show on our website and app funds our services. The generated revenue **helps us prepare new notes and improve the quality of existing study materials**, which are available on our website and mobile app.

If you don't use our website and app directly, it will hurt our revenue, and we might not be able to run the services and **have to close them**. So, it is a humble request for all to **stop sharing the study material** we provide on various apps. Please **share the website's URL instead.**

Subject Name: Soft Computing **Subject Code: IT 701**

Subject Notes

Syllabus:

Supervised Learning: Perceptron learning, Single layer/multilayer, Adaline, Madaline, Back propagation network, RBFN, Application of Neural network in forecasting, data compression and image compression.

Unit-2

Course Objectives

1. The objective of this course is to understand supervised learning and its type.
2. To help student to understand neural network work in forecasting and data compression, image compression .

Supervised Learning:

Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output.

$$Y = f(X)$$

The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data.

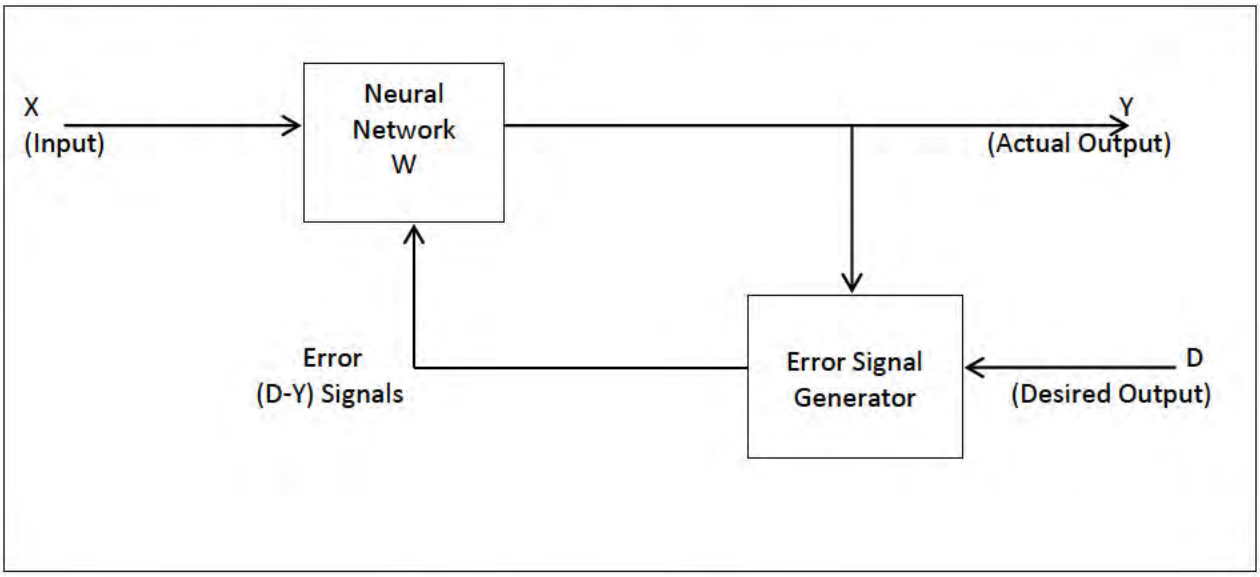

Figure 2.1: Supervised Learning

It is called supervised learning because the process of algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answers; the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance.

During training, the input vector is presented to the network, which results in an output vector. This output vector is the actual output vector. Then the actual output vector is compared with the desired (target) output vector. If there exists a difference between the two output vectors then an error signal is generated by the network. This error is used for adjustment of weights until the actual output matches the desired (target) output.

In this type of learning, a supervisor or teacher is required for error minimization. Hence, the network trained by this method is said to be using supervised training methodology. In supervised learning it is assumed that the correct "target" output values are known for each input pattern.

Perceptron Learning:

The perceptron is an algorithm for supervised learning of binary classifier. It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector. Perceptron network come under single-layer feed-forward networks.

The key points to be noted:

The perceptron network consists of three units, namely, sensory unit (input unit), associator unit (hidden unit), and response unit (output unit).

The sensory unit are connected to associator units with fixed weights having values 1, 0, or -1, which are assigned at random.

The binary activation function is used in sensory unit and associator unit.

The response unit has an activation function of 1, 0, or -1. The binary step with fixed threshold θ is sued as activation for associator. The output signals that are sent from the associator unit to the response time are only binary.

The output of the perceptron network is given by $y=f(y_{in})$

Where $f(y_{in})$ is activation function and is defined as:

$f(y_{in})$	=	1	If $y_{in} > \theta$
		0	If $-\theta <= y_{in} <= \theta$
		-1	If $y_{in} < -\theta$

Table 2.1: Perceptron Network

Perceptron Learning Rule:

In case of the perceptron learning rule, the learning signal is the difference between the desired and actual response of a neuron.

Consider a finite "n" number of input training vectors, with their associated target (desired) values x(n) and t(n), where "n" ranges from 1 to N. The target is either +1 or -1. The output "y"

is obtained on the basis of the net input calculated and activation function being applied over the net input.

$f(y_{in})$	=	1	If $y_{in} > \theta$
		0	If $-\theta <= y_{in} <= \theta$
		-1	If $y_{in} < -\theta$

Table 2.2: Perceptron Learning Rule

In original perceptron network, the output obtained from the associator unit is a binary vector, and hence the output can be taken as input signal to the response unit, and classification can be performed. Here only weights between the associator unit and the output unit can be adjusted, and the weights between the sensory and associator units are fixed.

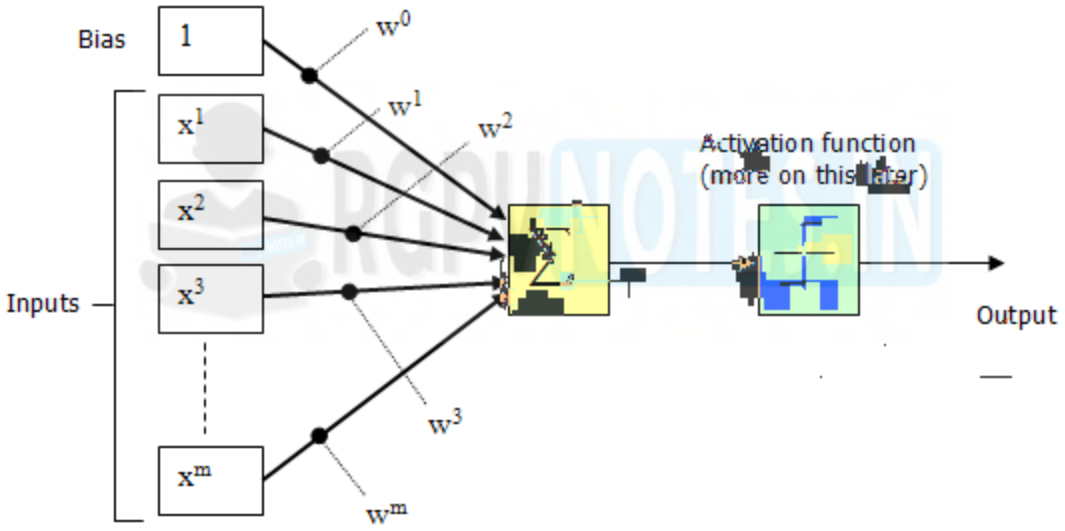

Figure 2.2: Perceptron Network

In above figure, there are n input neurons, 1 output neuron and a bias. The input-layer and output-layer neurons are connected through a directed communication link, which is associated with weights. The goal of the perceptron net is to classify the input pattern as a member or not a member to a particular class.

Perceptron Training Algorithm for Single Output Class:

The perceptron algorithm can be used for either binary input vectors, having bipolar targets, threshold being fixed or variable bias. In the algorithm, initially the inputs are assigned, then the net input is calculated. The output of the network is obtained by applying the activation function over the calculated net input. On performing comparison over the calculated and the

desired output, the weight updation process is carried out. The entire network is trained based on the mentioned stopping criteria.
The algorithm of a perceptron network is as follows:
Step 0: Initialize the weights and the bias (for easy calculation they can be set to zero). Also initialize the learning rate α (0<α<=1). For simplicity α is set to 1.

Step 1: Perform steps 2-6 until the final stopping condition is false.
Step 2: Perform steps 3-5 for each training pair indicated by s:t.
Step 3: The input layer containing input units is applied with identity activation function:
$x_i = s_i$
Step 4: Calculate the output of the network. To do so, first obtain the net input:

$$Y_{in} = b + \sum_{i=1}^{n} x_i w_i$$

where "n" is the number of input neurons in the input layer. Then apply activation over the net input calculated to obtain the output:

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > \theta \\ 0 & \text{if } -\theta <= y_{in} <= \theta \\ -1 & \text{if } y_{in} < -\theta \end{cases}$$

Step 5: Weight and bias adjustment: Compare the value of the actual (calculated) output and desired (target) output.
If y != t, then
$w_i(new) = w_i(old) + \alpha t x_i$
$b(new) = b(old) + \alpha t$
else, we have
$w_i(new) = w_i(old)$
$b(new) = b(old)$
Step 6: Train the network until there is no weight change. This is the stopping condition for the network. If this condition is not met, then start again form step 2.

Perceptron Training Algorithm for Multiple Output Class:
For multiple output classes, the perceptron training algorithm is as follows:
Step 0: Initialize the weights, biases and learning rate suitably.
Step 1: Check for stopping condition; if it is false, perform step 2-6.
Step 2: Perform step 3-5 for each bipolar or binary training vector pair s:t.
Step 3: Set activation (identity) of each input unit i=1 to n:
Xi = si
Step 4: Calculate output response of each output unit j=1 to m: First, the net input is calculated as

$$Y_{inj} = b_j + \sum_{i=1}^{n} x_i w_{ij}$$

The activation are applied over the net input to calculate the output response:

$$y_j = f(y_{inj}) = \begin{cases} 1 & \text{if } y_{inj} > \theta \\ 0 & \text{if } -\theta <= y_{inj} <= \theta \\ -1 & \text{if } y_{inj} < -\theta \end{cases}$$

Step 5: Make adjustment in weight and bias for j=1 to m and i=1 to n
If t_j != y_j, then
w_{ij} (new) = w_{ij} (old) + $\alpha t_j x_i$
b_j (new) = b_j (old) + αt_j
else, we have
w_{ij} (new) = w_{ij} (old)
b_j (new) = b_j (old)
Step 6: Test for the stopping condition, if there is no change in weights then stop the training process, else start again from step 2.

Single Layer / Multilayer Perceptron
Single layer perceptron is the first proposed neural model created. The content of the local memory of the neuron consists of a vector of weights. The computation of a single layer perceptron is performed over the calculation of sum of the input vector each with the value multiplied by corresponding element of vector of the weights. The value which is displayed in the output will be the input of an activation function.
In the Multilayer perceptron, there can more than one linear layer (combinations of neurons). If we take the simple example the three-layer network, first layer will be the input layer and last will be output layer and middle layer will be called hidden layer. We feed our input data into the input layer and take the output from the output layer. We can increase the number of the hidden layer as much as we want, to make the model more complex according to our task.

Linear Separability:
Consider two-input patterns (X1, X2) being classified into two classes as shown in figure. Each point with either symbol of x or o represents a pattern with a set of values (X1, X2). Each pattern is classified into one of two classes. Notice that these classes can be separated with a single line L. They are known as linearly separable patterns. Linear separability refers to the fact that classes of patterns with n-dimensional vector {x} = (x1, x2, …, xn) can be separated with a single decision surface. In the case above, the line L represents the decision surface.

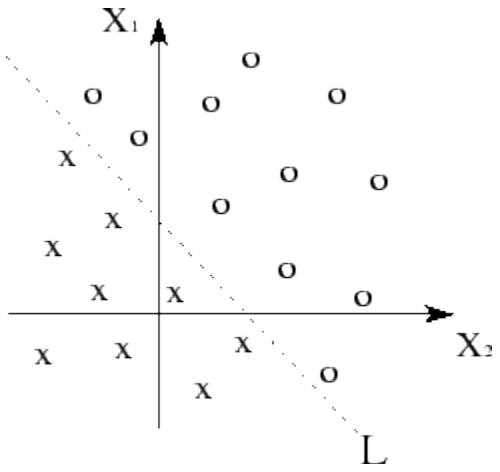

Figure 2.3: Linear Separability

The processing unit of a single-layer perceptron network is able to categorize a set of patterns into two classes as the linear threshold function defines their linear separability. Conversely, the two classes must be linearly separable in order for the perceptron network to function correctly. Indeed, this is the main limitation of a single-layer perceptron network.

The most classic example of linearly inseparable pattern is a logical exclusive-OR (XOR) function; is the illustration of XOR function that two classes, 0 for black dot and 1 for white dot, cannot be separated with a single line. The solution seems that patterns of (X1,X2) can be logically classified with two lines L1 and L2.

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

$Y = X_1 \oplus X_2$

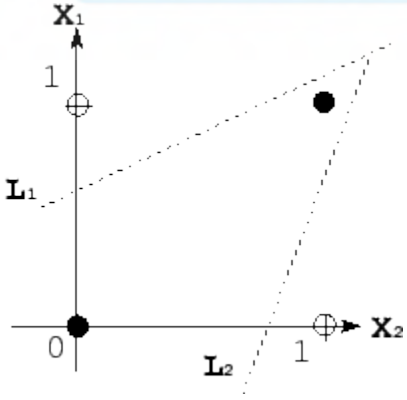

Figure 2.4: Example of Linear Separability

Adaptive Linear Neuron (Adaline):

The units with linear activation function are called linear units. A network with a single linear unit is called an Adaline. In an Adaline, the input-output relationship is linear. Adaline uses bipolar activation for its input signals and its target output. The Adaline network may be trained using delta rule. The delta rule also be called as least mean square (LMS) rule or Widrow-Hoff rule.

Architecture of Adaline:

The basic Adaline model consists of trainable weights. Inputs are either of the two values (+1 or -1) and the weights have sign (positive or negative). Initially, random weights are assigned. The net input calculated is applied to a quantizer transfer function that restores the output to +1 or -1. The Adaline model compares the actual output with the target output and on the basis of the training algorithm, the weights are adjusted.

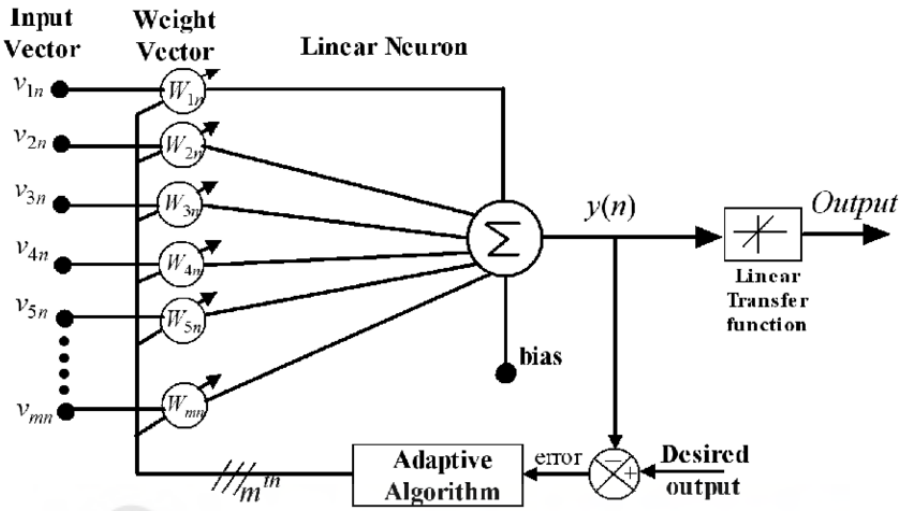

Figure 2.5: Architecture of Adaline

Training Algorithm:

The Adaline network training algorithm is as follows:

Step0: weights and bias are to be set to some random values but not zero. Set the learning rate parameter α.

Step1: Perform steps 2-6 when stopping condition is false.

Step2: Perform steps 3-5 for each bipolar training pair s:t

Step3: Set activations foe input units i=1 to n.

Step4: Calculate the net input to the output unit.

Step5: Update the weight and bias for i=1 to n

Step6: If the highest weight change that occurred during training is smaller than a specified tolerance then stop the training process, else continue. This is the test for the stopping condition of a network.

Testing Algorithm:

It is very essential to perform the testing of a network that has been trained. When the training has been completed, the Adaline can be used to classify input patterns. A step function is used to test the performance of the network. The testing procedure for the Adaline network is as follows:

Step 0: Initialize the weights. (The weights are obtained from the training algorithm.)

Step 1: Perform steps 2-4 for each bipolar input vector x.

Step 2: Set the activations of the input units to x.

Step 3: Calculate the net input to the output units

Step 4: Apply the activation function over the net input calculated.

Multiple Adaptive Linear Neurons (Madaline):

It consists of many Adaline in parallel with a single output unit whose value is based on certain selection rules. It uses the majority vote rule. On using this rule, the output unit would have an answer either true or false. On the other hand, if AND rule is used, the output is true if and only if both the inputs are true and so on. The training process of Madaline is similar to that of Adaline

Architecture:

It consists of "n" units of input layer and "m" units of Adaline layer and "1"Unit of the Madaline layer. Each neuron in the Adaline and Madaline layers has a bias of excitation "1". The Adaline layer is present between the input layer and the Madaline layer; the Adaline layer is considered as the hidden layer. The use of hidden layer gives the net computational capability which is not found in the single-layer nets, but this complicates the training process to some extent.

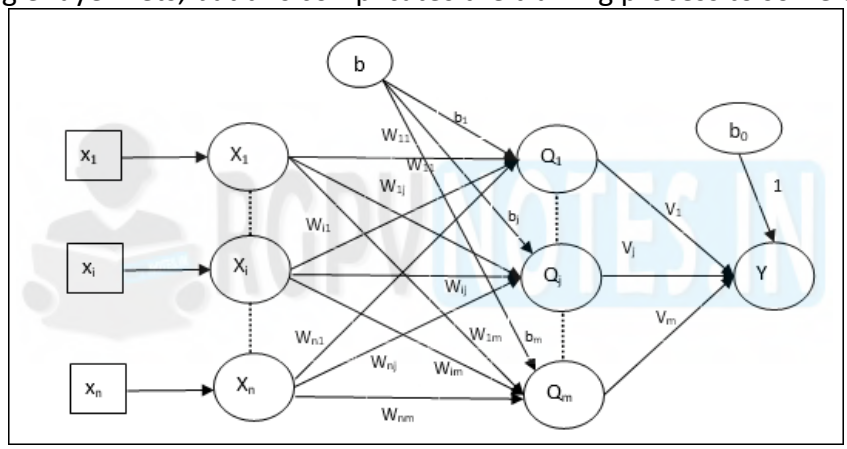

Figure 2.6: Architecture of Madaline

Training Algorithm:

In this training algorithm, only the weights between the hidden layers are adjusted, and the weights for the output units are fixed. The weights v1, v2……vn and the bias b0 that enter into output unit Y are determined so that the response of unit Y is 1.

Thus the weights entering Y unit may be taken as

$$v1=v2=….=vn=1/2$$

And the bias can be taken as

$$b_0 = 1/2$$

Step 0: Initialize the weights. The weights entering the output unit are set as above. Set initial small random valued for Adaline weights. Also set initial learning rate α.

Step 1: When stopping condition is false, perform steps 2-3.

Step 2: For each bipolar training pair s:t, perform steps 3-7.

Step 3: Activate input layer units, For i=1 to n.

$$xi = si$$

Step 4: Calculate net input to each hidden Adaline unit:

$$Z_{inj} = b_j + \sum_{i=1}^{n} x_i w_{ij},\ j=1 \text{ to } m$$

Step 5: Calculate output of each hidden unit:

$$z_j = f(z_{inj})$$

Step 6: Find the output of the net:

$$y_{in} = b_0 + \sum_{j=1}^{m} z_j v_j$$

Step 7: Calculate the error and update the weights.

1. If t = y, no weight updation is required.
2. If t != y and t =+1, update weights in zj, where net input is closest to 0 (zero):

$$b_j(new) = {}_{bj}(old) + \alpha\,(1-z_{inj})$$

$$w_{ij}(new) = w_{ij}(old) + \alpha\,(1-z_{inj})x_i$$

3. If t != y and t =-1, update weights on units zk whose net input is positive:

$$w_{ik}(new) = w_{ik}(old) + \alpha(-1-z_{ink})x_i$$

$$b_k(new) = b_k(old) + \alpha(-1-z_{ink})$$

Step 8: Test for the stopping condition. (If there is no weight change or weight reaches a satisfactory level, or if a specified maximum number of iterations of weights updation have been performed then stop, or else continue).

Back Propagation Network:

Back propagation learning algorithm is one of the most important developments in neural networks. This learning algorithm is applied to multilayer feed-forward networks consisting of processing elements with continuous differentiable activation functions. The network associated with back propagation learning algorithm is called back-propagation networks (BPNs).

Architecture:

A back propagation neural network is a multilayer, feed-forward neural network consisting of an input layer, a hidden layer and an output layer. The neurons present in the hidden and output layers have biases, which are the connections from the units whose activation is always 1. The bias terms also acts as weights.

The inputs are sent to the BPN and the output obtained from the net could be either binary (0,1) or bipolar (-1, +1). The activation function could be any function which increases monotonically and is also differentiable.

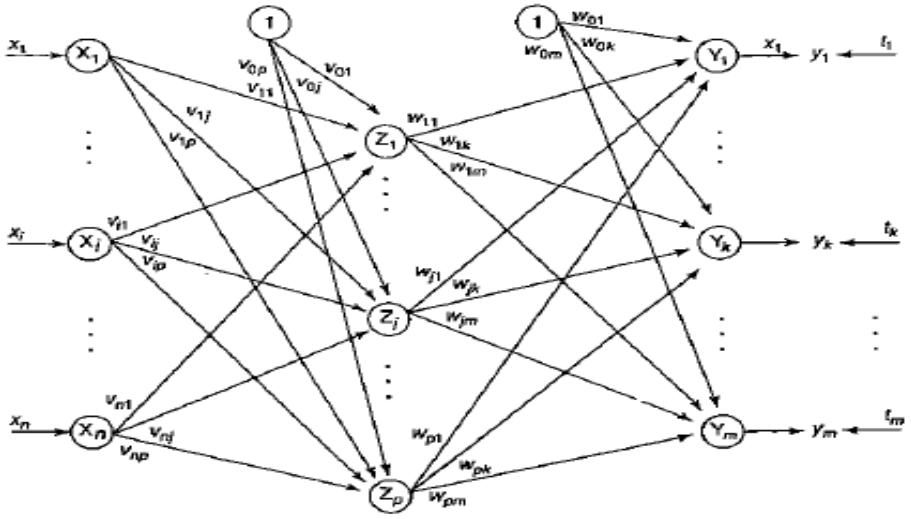

Figure 2.7: Architecture of Back Propagation Network

Training Algorithm:

Step 0: Initialize weights and learning rate (take some small random values).

Step 1: Perform Steps 2-9 when stopping condition is false.

Step 2: Perform Steps 3-8 for training pair.

Feed-forward phase (Phase-I):

Step 3: Each input unit receives input signal x; and sends it to the hidden unit (i = l to n}.

Step 4: Each hidden unit Zj (j = 1 to p) sums its weighted input signals to calculate net input:

$$z_{inj} = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}$$

Calculate output of the hidden unit by applying its activation functions over Zinj (binary or bipolar sigmoidal activation function}: -

$$z_j = f(z_{inj})$$

and send the output signal from the hidden unit to the input of output layer units.

Step 5: For each output unity_k(k = I to m), calculate the net input:

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

and apply the activation function to compute output signal

$$y_k = f(y_{ink})$$

Back-propagation of error (Phase-II):

Step 6: Each output unit y_k(k = I to m) receives a target pattern corresponding to the input training pattern and computes the error correction term:

$$\delta_k = (t_k - y_k) f'(y_{ink})$$

The derivative $f(y_{ink})$ can be calculated. On the basis of the calculated error correction term, update the change in weights and bias:

$$\Delta w_{jk} = \alpha \delta_k z_j;$$

$$\Delta w_{0k} = \alpha\delta_k;$$

Also, send δ_k to the hidden layer backwards.

Step 7: Each hidden unit (zj,j = I top) sums its delta inputs from the output units:

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

The term δ_{inj} gets multiplied with the derivative of $f(Z_{inj})$ to calculate the error term:

$$\delta_j = \delta_{inj}\, f'(z_{inj})$$

The derivative $f'(z_{inj})$ can be calculated. Depending on whether binary or bipolar sigmoidal function is used. On the basis of the calculated δ_j, update the change in weights and bias:

$$\Delta v_{ij} = \alpha\delta_j x_i;$$

$$\Delta v_{0j} = \alpha\delta_j;$$

Weight and bias updation (Phase-III):

Step 8: Each output unit (yk, k=1 to m) updates the bias and weights:

$$w_{jk}(new) = w_{jk}(old) + \Delta w_{jk}$$

$$w_{ok}(new) = w_{ok}(old) + \Delta w_{ok}$$

Each hidden unit (zj, j= 1 to p) updates its bias and weights:

$$v_{ij}(new) = v_{ij}(old) + \Delta v_{ij}$$

$$v_{oj}(new) = v_{oj}(old) + \Delta v_{oj}$$

Step 9: Check for the stopping condition. The stopping condition may be certain number of epochs reached or when the actual output equals the target output.

Radial Basis Function Network:

The radial basis function (RBF) is a classification and functional approximation neural network developed

by M.J.D. Powell. The network uses the most common nonlinearities such as sigmoidal and Gaussian kernel functions. The Gaussian functions are also used in regularization networks. The response of such a function is positive for all values of y; the response decreases to 0 as lyl→ 0. The Gaussian function is generally defined as

$$f(y) = e^{-y2}$$

The derivative of this function is given by:

$$f'(y) = -2ye^{-y2} = -2yf(y)$$

Architecture:

The architecture consist of two layers whose output nodes form a linear combination of the kernel (or basis) functions computed by means of the RBF nodes or hidden layer nodes. The basis function (nonlinearity) in the hidden layer produces a significant nonzero response w the input stimulus it has received only when the input of it falls within a small localized region of the input space. This network can also be called as localized receptive field network.

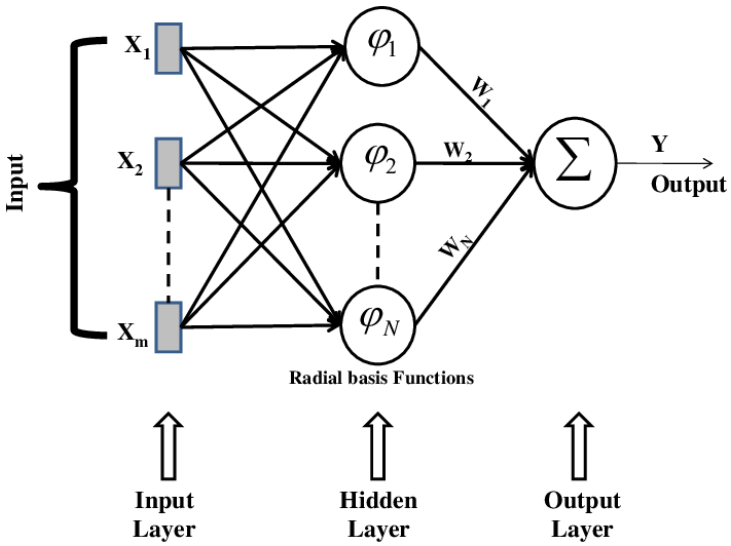

Figure 2.8: Architecture of RBF

Training Algorithm:

Step 0: Set the weight to small random values.

Step 1: Perform steps 2-8 when the stopping condition is false.

Step 2: Perform steps 3-7 for each input.

Step 3: Each input unit (xi for all I = 1 to n) receives input signals and transmits to the next hidden layer unit.

Step 4: Calculate the radial basis function.

Step 5: Select the centers for the radial basis function. The centers are selected from the set of input vectors. It should be noted that a sufficient number of centers have to be selected to ensure adequate sampling of the input vector space.

Step 6: Calculate the output from the hidden layer unit:

$$v_i(x_i) = \frac{\exp\left[-\sum_{j=1}^{r}(x_{ji} - \hat{x}_{ji})^2\right]}{\sigma_i^2}$$

^

where $x_{ji}$is the center of the RBF unit for input variable; $\sigma_i$the width of ith RBF unit; $x_{ji}$the jth variable of input pattern.

Step 7: Calculate the output of the neural network:

$$y_{net} = \sum_{i=1}^{k} w_{im} v_i(x_i) + w_0$$

where k is the number of hidden layer nodes (RBF function);y_{net} the output value of mth node in output layer for the nth incoming pattern; $W_{im}$the weight between ith RBF unit and mth output node; W_o the biasing term at nth output node.

Step 8: Calculate the error and test for the stopping condition. The stopping condition may be number of epochs or to a certain extent weight change.

Application of Neural Network:

Forecasting:

Load forecasting is very essential to the operation of electricity companies. It enhances the energy-efficient and reliable operation of a power system. The inputs used for the neural network are the previous hour load, previous day load, previous week load, day of the week, and hour of the day. The neural network used has 3 layers: an input, a hidden, and an output layer. The input layer has 5 neurons, the number of hidden layer neurons can be varied for the different performance of the network, while the output layer has a single neuron.

Many methods have been used for load forecasting in the past. These include statistical methods such as regression and similar-day approach, fuzzy logic, expert systems, support vector machines, econometric models, end-use models, etc. A supervised artificial neural network has also been used for the same. The neural network is trained on input data as well as the associated target values. The trained network can then make predictions based on the relationships learned during training.

Data Compression:

The transport of data across communication paths is an expensive process. Data compression provides an option for reducing the number of characters or bits in transmission. It has become increasingly important to most computer networks, as the volume of data traffic has begun to exceed their capacity for transmission. Artificial Neural Network (ANN) based techniques provide other means for the compression of data at the transmitting side and decompression at the receiving side. The security of the data can be obtained along the communication path as it is not in its original form on the communication line. Artificial Neural Networks have been applied to many problems, and have demonstrated their superiority over classical methods when dealing with noisy or incomplete data. One such application is for data compression. Neural networks seem to be well suited to this particular function, as they have an ability to preprocess input patterns to produce simpler patterns with fewer components. This compressed information (stored in a hidden layer) preserves the full information obtained from the external environment. The compressed features may then exit the network into the external environment in their original uncompressed form.

Image Compression:

Neural networks can accept a vast array of input at once, and process it quickly, they are useful in image compression.

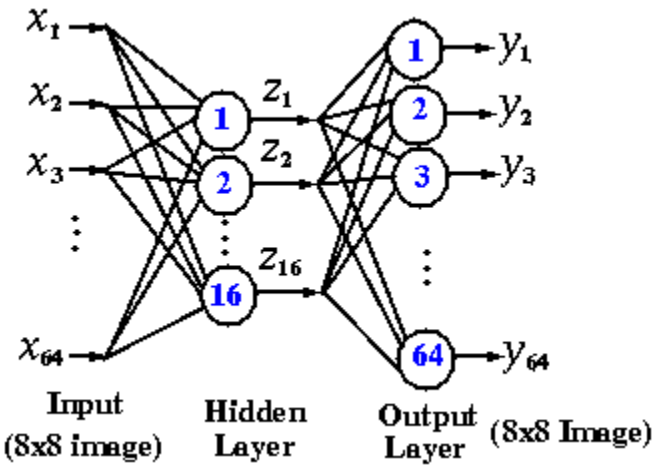

Figure 2.9: Bottleneck-type Neural Net Architecture for Image Compression

Here is a neural net architecture suitable for solving the image compression problem. This type of structure is referred to as a bottleneck type network, and consists of an input layer and an output layer of equal sizes, with an intermediate layer of smaller size in-between. The ratio of the size of the input layer to the size of the intermediate layer is - of course - the compression ratio.

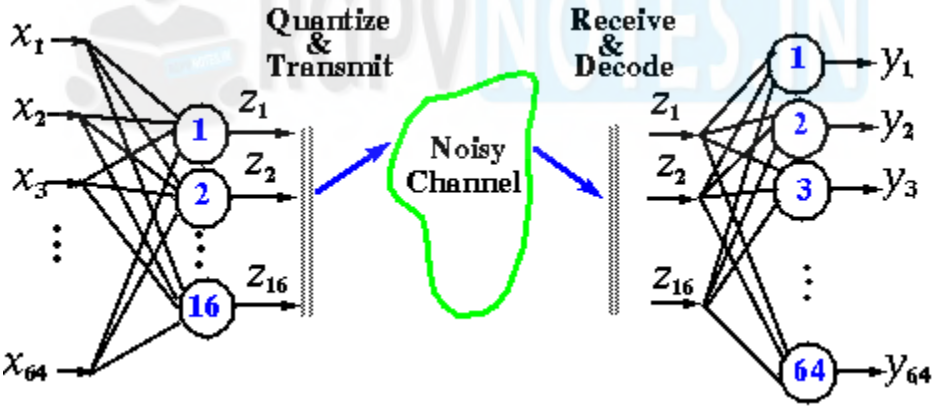

Figure 2.10: Bottleneck architecture for image compression over a network or over time

This is the same image compression scheme, but implemented over a network. The transmitter encodes and then transmits the output of the hidden layer, and the receiver receives and decodes the 16 hidden outputs to generate 64 outputs.

Pixels, which consist of 8 bits, are fed into each input node, and the hope is that the same pixels will be outputted after the compression has occurred. That is, we want the network to perform the identity function.

Actually, even though the bottleneck takes us from 64 nodes down to 16 nodes, no real compression has occurred. The 64 original inputs are 8-bit pixel values. The outputs of the hidden layer are, however, decimal values between -1 and 1. These decimal values can require a possibly infinite number of bits.

RGPVNOTES.IN

Thank you for using our services. Please support us so that we can improve further and help more people.
https://www.rgpvnotes.in/support-us

If you have questions or doubts, contact us on
WhatsApp at +91-8989595022 or by email at hey@rgpvnotes.in.

For frequent updates, you can follow us on
Instagram: https://www.instagram.com/rgpvnotes.in/.