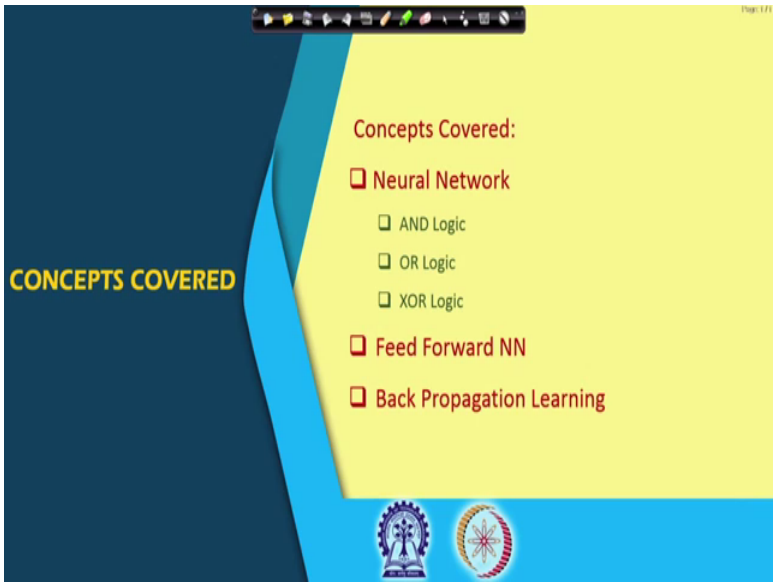
**Deep Learning**
**Prof. Prabir Kumar Biswas**
**Department of Electronics and Electrical Communication Engineering**
**Indian Institute of Technology, Kharagpur**

**Lecture - 21**
**Multilayer Perceptron**

Hello, welcome to the NPTEL online certification course on Deep Learning, we have started our discussion on the Neural Network.

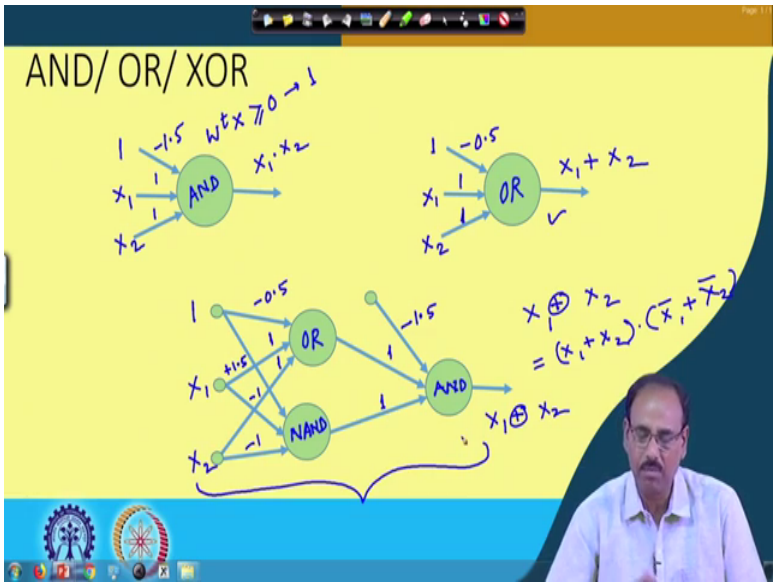
(Refer Slide Time: 00:37)




And, in our previous class we have talked about the basic implementation of few of the logic functions AND, OR and XOR function using the neural networks. Today we will continue our discussion with the neural network and particularly we will talk about the feed forward neural network also known as multilayer perceptron. And, then we will see that how these neural networks can be trained to solve certain problems.

So, for that the algorithm that we will talk about is what is known as back propagation learning algorithm. And in fact, we will talk about back propagation learning in details, there is not being that this back propagation learning is the basic of the deep learning or the deep neural networks, that we are going to discuss in future.

(Refer Slide Time: 01:37)




So, just to recapitulate what we did in the previous class is that we have implemented three of the logic functions AND function, OR function and XOR function using the neural network. And, we have seen that in case of an AND function, if the input is X 1 and X 2 for unified representation as we have done before, that we have appended an additional component which is equal to 1 and, this is what helps in giving a bias to the neural network or every nodes in the neural network.

And, the weights that we considered was minus 1.5, 1 and 1 and with these we have seen in the previous class that output becomes X 1 and X 2. And, that is what our AND function. In case of an OR function of course, there was a nonlinearity involved in it, the nonlinearity that we have considered was a threshold nonlinearity; that means, if the output or weighted sum of the input, that is W transpose X, this was greater than or equal to 0, we have assumed the output to be 1, and if it is less than 0 we have assumed the output to be 0. And, with that we have seen that this network the simple node implements AND function.

Similarly, in case of OR function again our input is the binary input X 1 and X 2, the bias term in this case is minus 0.5, here we have given the weights as 1 and 1 and with this again we have seen that the output becomes X 1 or X 2. We could implement this AND function and OR function using a single neural network, because we have seen before that these functions are actually linear functions. I can separate the outputs which are

once from the outputs which are 0s by a straight line, which has not possible in case of an XOR gate.

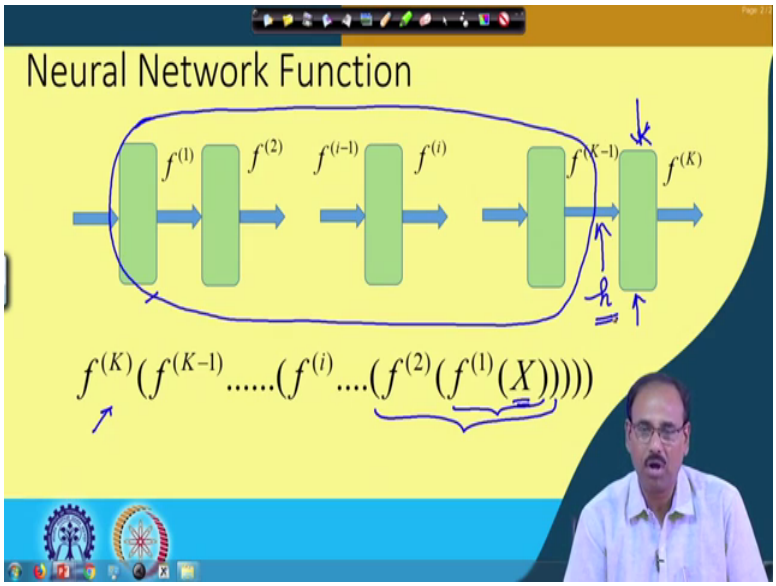
So, XOR is a non-linear function and we have seen in the previous class, that because XOR is a non-linear function. So, it cannot be implemented a single neural network or a single neuron. So, I need multi-layer neural network and that is what is in this figure. So, here again our inputs are 1, X 1 and X 2 those are the binary inputs and one of these two gates we have seen before that it implements an OR gate and the other one implements a NAND function.

So, let us assume that the first one implements and OR function and this one implements a NAND function. So, for implementing OR function just as we have seen over here or weights will be minus 0.5, 1 and 1 and for implementing a NAND function it has to be just complement of AND. So, weights for implementing NAND function will be plus 1.5 minus 1 and minus 1 and, these two outputs are finally to be ended. Because, our XOR function X 1, XOR, X 2 is nothing but X 1 or X 2 anded with X 1 NAND X 2.

So, here I have to have a AND function so, for this a weights will be minus 1.5, 1 and 1 so, here at the output what I get is X 1, XOR, X 2. So, you find that these are simple implementations of the logic functions using neural networks.

Now, from particularly this XOR function, it is quite obvious that if the function that we have to implement or the problem is a non-linear problem, the non-linear problem cannot be solved using a single NAND network. I need multiple or multilayer neural network for implementing a non-linear problem.

(Refer Slide Time: 06:46)




So, for a non-linear problem as we also have seen in the previous class, that I have layers of neural networks So, I have neural networks in multiple layers. And, from every layer say layer 1 to layer 2, I have complete connection that is every node in the layer 1 is connected to every node in the layer 2. And, if the final function so here you will find that there are K number of layers so, at the output I call the output layer to be Kth layer. So, the final function that is being implemented is f K ok and, the input layer implements a function f 1.

So, with my input vector at as X, the first layer that is f 1 layer, it implements function f 1 of X, the second layer implements f 2 of f 1 X, third layer will implement f 3 of f 2 of f 1 X. And, finally, the Kth layer implements f K of the output that has been generated by all the previous layers.

So, when it is a non-linear problem, we can see that all the layers from f 1 to f K minus 1, they will implement a non-linear mapping. Because, as we said earlier that if I have a non-linear problem, then instead of trying to design a non-linear classifier, you try to map the input vectors using a non-linear mapping function. So, that they are mapped into intermediate feature space and in the intermediate feature space, this non-linearly mapped input vectors will be linearly separable. And, then finally, at the final layer or at the Kth layer I can have a linear classifier to classify all those samples correctly.

So, all these layers from f 1 to f K they actually implement this non-linear mapping. So, at the output of f K, the new feature vectors that I gets a feature vectors X, which are now linearly separable And, the Kth layer I can implement a linear classifier which we will classify all these vectors h, which are now linearly separable.

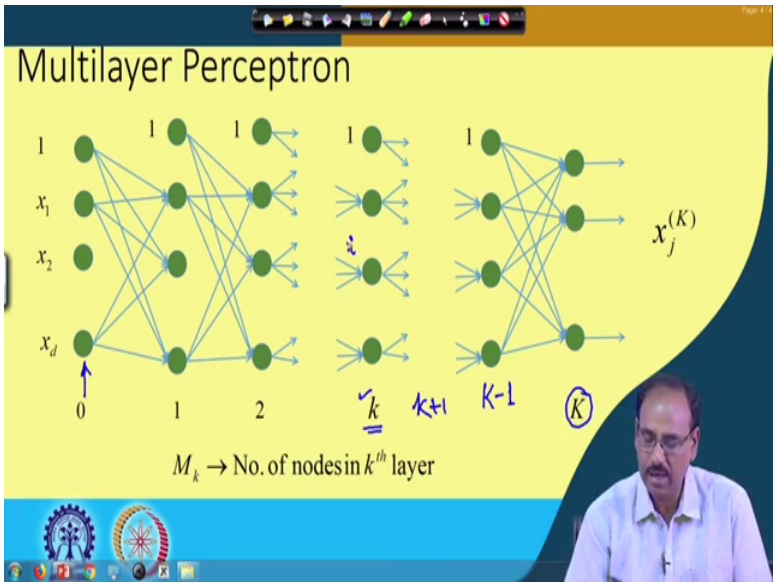
So, this is just a block diagram representation of a multilayer perceptron or a feed forward neural network. Now, why it is feed forward, because I am inputting the feature vector X at the input layer, they are being processed at every layer and being forwarded in the forward direction to the next layer and, finally, you get the output from the final layer, at the output layer.

And, no where in this path the information is fed back to the previous layer. So, always the information flows in the forward direction so, it is feed forward network. But we will see later that for learning or for training this neutral network, the error is propagated in the backward direction, because our aim is to minimize the error by adjusting the weights in between the layers.

So, for training this neural network the error is propagated in the backward direction through each of the layers and while it is being propagated at every layer the weights are updated in order to minimize the error. So, that is why the learning algorithm is known as back propagation learning. So, our neural network is a feed forward neural network, but in the learning algorithm is a back propagation learning algorithm.

Now, let us see in details that this how this neural network or multilayer perceptron that looks like.

(Refer Slide Time: 10:50)

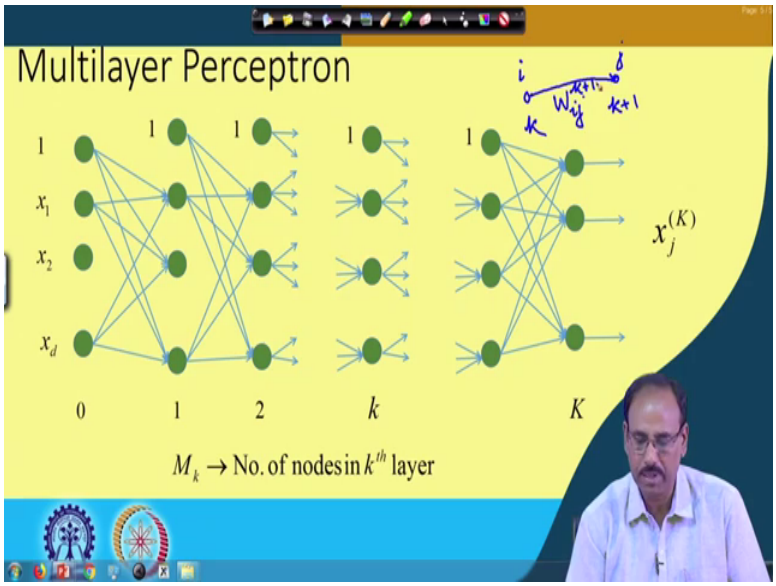



So, this is what is a somewhat detailed representation of this feed forward neural network. So, here I have assumed that there are K number of layers in the network; at the beginning we have an input layer which is represented as 0th layer. The purpose of this layer is simply whatever comes at the input is simply passes to the output. So, this layer does not have any other function, other than simply passing the input to the output. And, every other layer from 1 2 see K minus 1, each of these layers participate in non-linearly mapping the input feature vector x to a new feature space h.

And, Kth layer which is represented by capital K, it is the final output layer right. And, we also assume that from every layer every intermediate layer k, where this k it is represented by largest later, the nodes are connected or the neurons are connected to the next layer which is k plus 1.

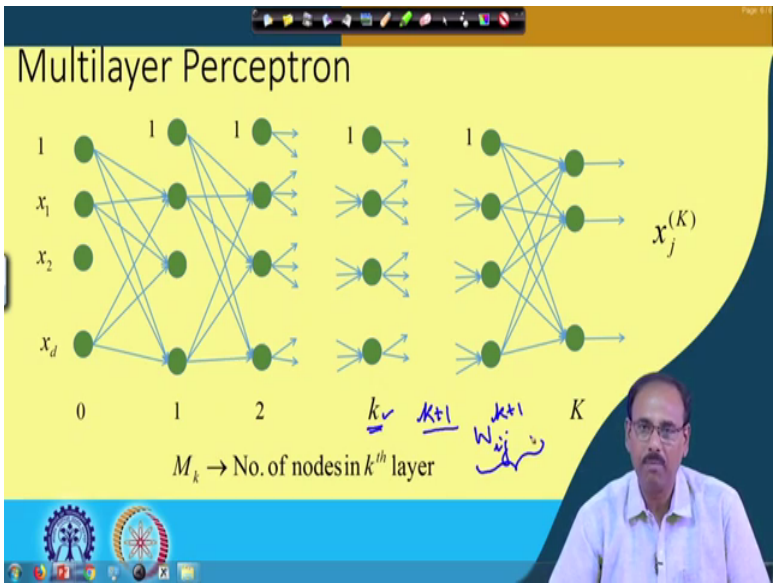
And this connection is complete in the sense, that every node in the Kth layer is connected to every node in the k plus first layer. And, if I take an ith node in the Kth layer it is connected to jth node in the k plus first layer through so let me put it like this.

(Refer Slide Time: 12:27)




So, I take a node i or a neuron i in the kth layer and I take a neuron j in the k plus first layer, which is the next layer. So, the output from the node i is connected to the input of node j output from node i in kth layer is connected to the input of k plus first layer through a connection weight, which is W ij k plus 1. So, this is the convention that we will use when we discuss about the back propagation learning.

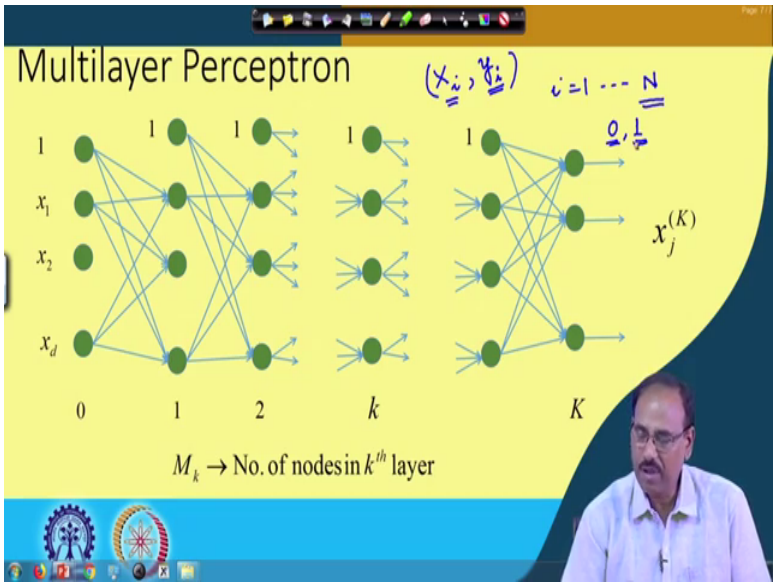
(Refer Slide Time: 13:24)




So, this is the overall architecture of our feed forward neural network there are K number of layers, Kth layer is the final output layer. Every node or every neuron i in an

intermediate layer k represented by lower cost k is connected to the next connected to the jth node in the k plus first layer through an weight which is given by W i j k plus 1. And, the purpose of training this neural network or learning is that iteratively using the training vectors, you try to find out what should be the value of W ij k plus 1 for every i j and k. So, that the neural network is finally trained to solve your problem. So, that is the purpose of back propagation neural network.

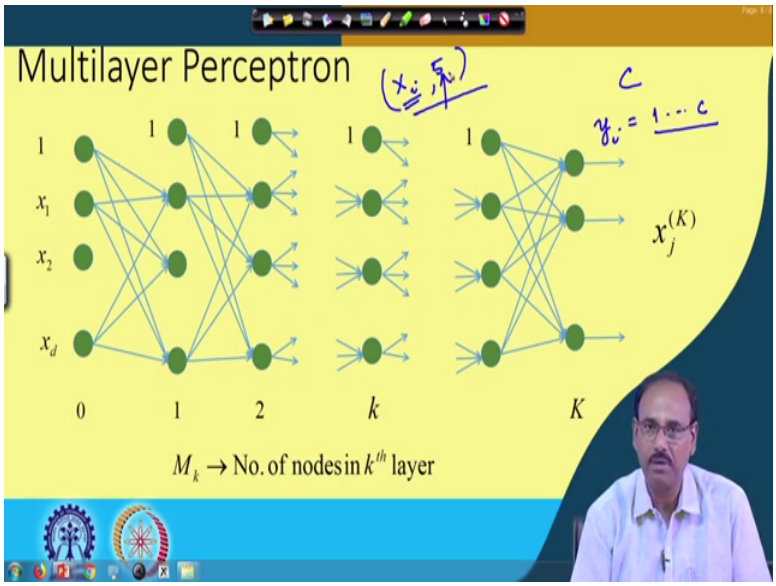
(Refer Slide Time: 14:17)




And, in this network you find that as we are feeding the input x i, so for training, the training data is fed in the form of X i y i as a doublet for say i is equal to 1 to N, where N is the number of training samples I have, which is given for training this neural network.

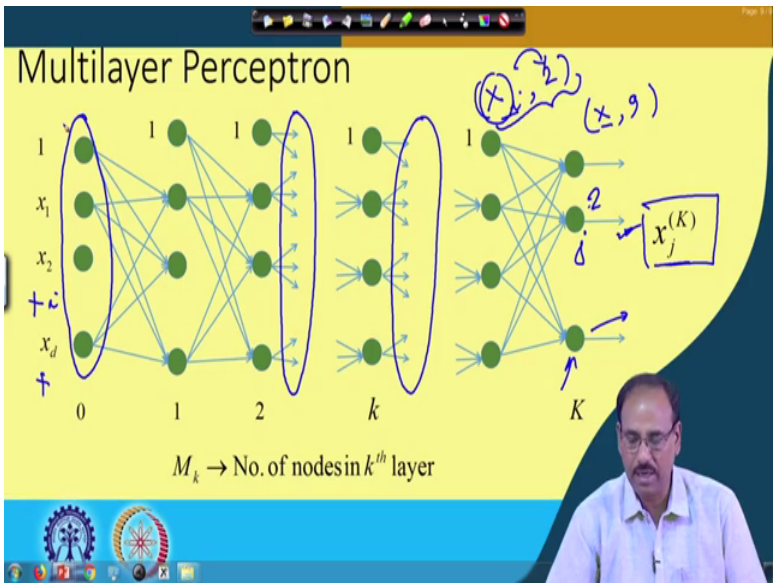
So, for every training sample X i as it is labeled because these are used for the training purpose, I know that to which class this sample X i belongs. So, X i in this case belongs to class y i. So, if I have a binary classification problem we will do that quickly this y i can be either 0 or 1. If, y i is 0; that means, the sample X i belongs to say class 1, omega 1 and if y i is 1; that means, the sample X i belongs to class omega 2. So, this tells you that what is the class belongingness of the training sample that I have.

(Refer Slide Time: 15:29)




If, I have C number of classes, if I have C number of classes, then y i will take 1 of the values 1 to c. So, this y i in that case is the index to the class to which X i belongs. So, if I have a data X i 5, this is the training data given, then this means that the training data X i belongs to class 5.

(Refer Slide Time: 16:08)




Secondly at the output node, if I take a jth node; jth neuron at the output layer I represent the output of the jth node in the Kth layer as x j K. So, these are the conventions that we will use, when we talk about back propagation learning. And, here you find that only at

the output given a training vector X i only I know that what should be the corresponding output. Because, if it is x i 2 this is the training pair that is given I know that when this X i is fed to the input output of the second node that should be high. The outputs of all other nodes should be equal to 0 because, my training pair says that this vector X i belongs to class 2.

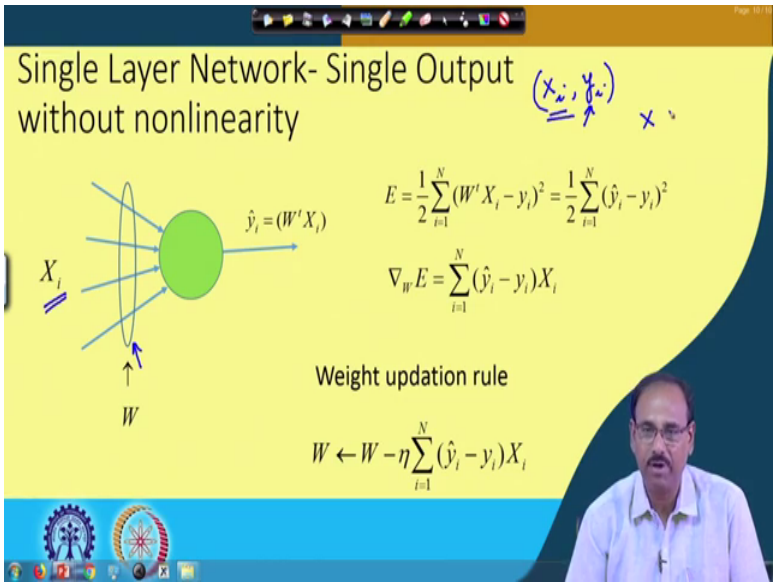
Similarly, if attaining vector X is given which belongs to class 9, then when I feed this X to the input only the output of the 9th node from the output layer should be high and all other outputs should be low. So, this I can decide only at the output layer. I really do not know that what should be outputs of any of the hidden layers, that is not visible. So, that is the reason that all the nodes or all the layers, except the output layer, they are known as hidden layers. Because, I can only observe I can only decide at the output I cannot decide what should be the outputs of any node in the intermediate layers or hidden layers.

And, this is a layer as we said that this is known as input layer. So, the purpose of every neuron in the input layer is simply to pass whatever is coming to the input to it is output. And, it is subsequently fed to the neurons of the next layer. So, this is what the architecture of the neural network looks like and the conventions that will follow.

(Refer Slide Time: 18:19)


Now, let us see that how can we train the neural network or what is back propagation learning.

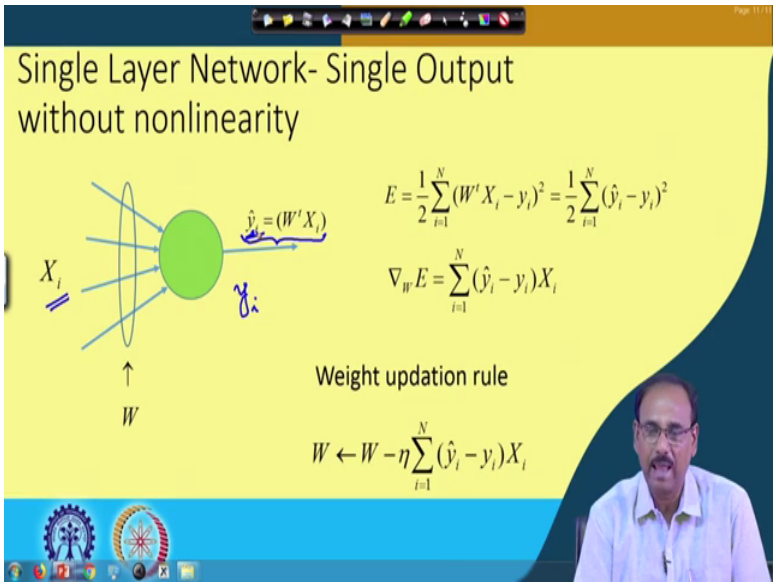
(Refer Slide Time: 18:25)




So, to talk about back propagation learning first I will consider a very simple neural network consisting of a single layer which is the output layer. Of course, you remember that we said before that using a single layer, I can solve only linear problems; I cannot solve any non-linear problem. And, as I said that this learning algorithms will do in details, because this forms the basis of all subsequent deep learning or deep networks deep neural networks that we will talk about. So, it is very important that you understand the back propagation learning very very clearly right.

So, again I take a single network a single neuron, where the weight vector is W, if I feed an input vector X i. So, as we said before that for training I get the input vectors as pairs X i y i, where this y i is the class index to which this vector X i belongs. And, as we said that this is used only for training purpose. During actual testing or when you deploy such neural networks, I have an unknown vector X i do not know what is the y; if, I know then I do not have to classify that and that classification will be done by the neural network only.

(Refer Slide Time: 20:02)




So, here as I have or as I feed such an input vector X i, I expect that the output of the neural network should be y i and, only when I get the output as y i, which matches with my true values. Then, at least this X i is correctly classified by this neuron, but what we get is I get a value X i hat, which is an approximation of X i.

So, if X i hat so this X i hat is nothing, but W transpose X i, where W is my weight vector and X i is the input vector. And, for the time being I am assuming that I have our neural network with single layer and single output node and I am not assuming, there is any nonlinearity in this neuron ok.

So, my output simply becomes W transpose X i and I am assuming that this W transpose X i is an approximation to y i which is y i hat. So, if y i and y i hat there same; that means, my input vector is correctly classified. So, I do not have to take any action to modify the weight vector W. Now, suppose y i hat and y i they are not same they are different, ok.

So, my error will be y hat minus y i this is the error and what I compute is the sum of squared error. So, this is sum of y i minus X i where this I will vary from 0 to N so, this squared error is computed over all the training vectors that I have as I varying from 0 to capital N and I have capital N number of vectors for training purpose.

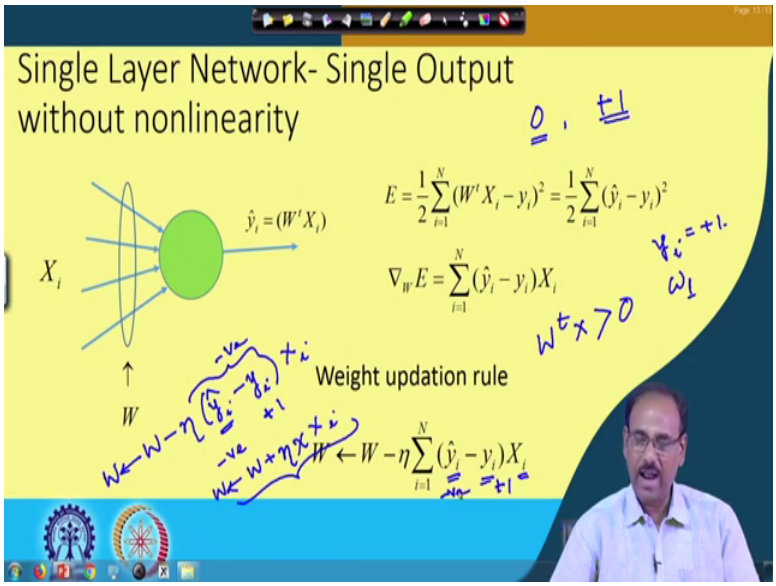
And, I scaled it this by half the reason of scaling it by half is as we have seen earlier also for any learning algorithm or any training algorithm, we go for gradient descent approach; that means, I have to take differentiation of the error function or the loss function that we have generated. And, because it is squared error or squared loss so having a factor half will simplify our expressions; so, that is the reason this half is put. If, I put it in an elaborated form so, this error function of the loss function E is nothing, but half of W transpose X is this W transpose X i is nothing, but our y hat. So, W transpose X i minus y i squared take the summation over all I varying from i equal to 1 to N; that means, you are summing the errors of all the training vectors that you have.

Next as we said that we will employ the gradient descent approach as we have done before for training the network or for updating the weight vector W. So, I take the gradient of E with respect to weight vector W. And, here you find that if this gradient is nothing, but y hat minus y i into X i where X i is the input training vector, right.

And, now we find that y we have put this half, because otherwise there would have been a scaling function 2 over here a scaling vector 2 over here so, in order to avoid that you put half. So, this is the gradient of the error or the loss right. So, I have to update W or the weight vector in such a way, that this loss is minimized or the error is minimized. And, for that as before my weight updation rule follows the gradient descent procedure. So, my weight updation will be simply W gates, if this is the previous value of the weight vector the updated weight vector will be W minus some eta times the gradient. What is this eta? We also said before that this is nothing, but a rate of convergence factor.

So, this eta indicates if the value of eta is very high, then the rate of convergence will be fast, if the value of eta is low, then the rate of convergence will be low. So, I will have a slower convergence; of course, both has their individual merits and demerits we also discussed about those before.

(Refer Slide Time: 24:56)




Now, if you look at this weight updation rule, you find that we get something more. I said my output y should be either 0 or plus 1. So, if it is 0 it belongs to 1 class, if it is plus 1 it belongs to another class. And, if you remember that earlier the linear discriminators that we have talked about, that is a linear plane which separates two different classes which are linearly separable. We have said for one of them, if W transpose X is greater than 0, then it belongs to 1 class if it is less than 0 it belongs to 1 another class.

Now, here let us come to a situation that, if I find that W transpose X is greater than 0 for samples belonging to sum class omega 1, and this is represented by y i is equal to plus 1. So, as long as my W transpose X i is greater than 0, then it is correctly classified, right.

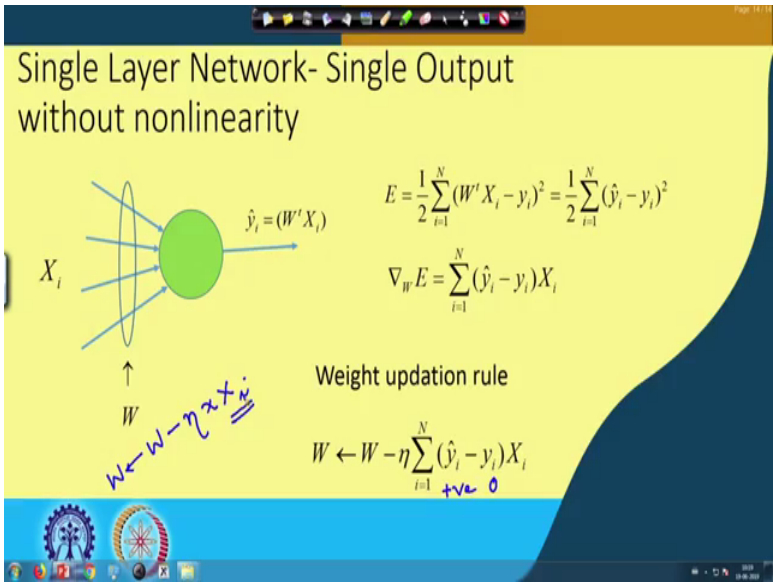
Now, here you find that if my y i is plus 1, but I get y i hat to be negative, then and instead of taking the sum let us consider a single feature vector X i. So, which is nothing, but stochastic optimization we have also talked about that before. If you sum all of them; that means, you are considering all the training vectors together it becomes a batch optimization technique. So, you have talked about batch optimization meaning of the mini batch optimization and stochastic optimization.

So, if I consider only X i that becomes a stochastic optimization procedure and then our weight updation rule will be simply W gates W minus eta times y i hat minus y i times X i. So, here if my y i is actually plus 1; that means, W transpose X should be greater than

0 for correct classification of X i, but suppose W transpose X which is nothing but y i hat happens to be negative. In that case this y i hat minus y i this whole term will be negative. And, in effect what we are doing is we are making W updating W as W plus some factor say eta times let us put chi times X i.

So, here you find that there is some similarity with the perceptron algorithm that we talked about towards the beginning of our course. That, if an X i is misclassified we add a fraction of X i to the weight vector for weight vector updation, right. In the same manner, if I assume that y i is 0 for which this y i hat should be negative, because it belongs to the other class, but if I get y i to be positive y i hat to be positive.

(Refer Slide Time: 28:32)




So, over here my y i is 0; that means, X i belongs to another class, for which y i hat which is computed by this neuron should be negative, but if I get this as positive ok. Again as before, the updation that I am doing is W as W minus some eta chi times X i.

So, this is the other vector for which W transpose X i should have been negative, but it has been misclassified because y i hat has been positive has been computed as positive by the neuron by the neural network. And, in this case what we are doing is we are subtracting a fraction of y i from W or in the other words we are adding a fraction of negated y i to W, again the same thing that we have done in case of our perceptron algorithm.

So, this is what you have in case of single layer perceptron that following gradient descent procedure the way we update the weights or the weight vector is by adding or subtracting a fraction of the misclassified samples to the weight vectors. So, I will stop this lecture here we will continue with this next.

Thank you.