Deep Learning

Prof. Prabir Kumar Biswas
Department Of Electronics And Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 22
Multilayer Perceptron – II

Hello, welcome to the NPTEL online certification course on Deep Learning. We are discussing about the Multilayer Perceptron or feed forward neural network.

(Refer Slide Time: 00:41)

CONCEPTS COVERED

Concepts Covered:

- ❑ Neural Network
- ❑ Feed Forward NN
- ❑ Back Propagation Learning

So, in your previous class we have given an introduction to the multilayer perceptron or feed forward neural network. And we have also started our discussion on training a neural network or the algorithm, which is known as back propagation learning.

So, in the previous class we have considered a back propagation learning algorithm or back propagation training procedure, for a single their neural network having only one neuron that is a function which is having only one output. And while discussing that, we also have assumed that the neuron does not impose any nonlinearity. That means, given the training vector X and W, being the weight vector of the neural network the output simply becomes W transpose X. And because the neural network had only one output; so, the output could be either positive or negative.

So, if it is positive then the sample input vector is classified to one class; if it is negative, it is classified to another class. And we considered the case that the training vectors are given as pair ordered pair, given in the form X i y i where X i is the training vector and y i indicates the index to the class to which the X i belongs. And because it was a two class problem so, we assumed that y i could assume either of value 0 or a value 1.

Whereas, W transpose X when I compute W being the weight vector and X being the feature vector or the input vector, it is not necessary that W transpose X will always be either 1 on 0 or 0. In fact, it will be a real number it may be 0; it may be greater than 0, it may be less than 0. So, a classification rule was that if it is greater than 0 it is belonging to one class, if it is less than 0 it belongs to another class.

But coming to the neural network, the output should be either 1 or 0; that means, if W transpose X is greater than 0; the output should be 1 indicating that it belongs to class the omega 1. If the output is less than 0, then it should be truncated to 0 indicating that it belongs to another class. So, that our class index 0 and 1, matches with whatever you get from the as output of the neuron. And in order to do that we also have seen before that when we have an implemented an OR function or AND function or XOR function with the help of neurons.

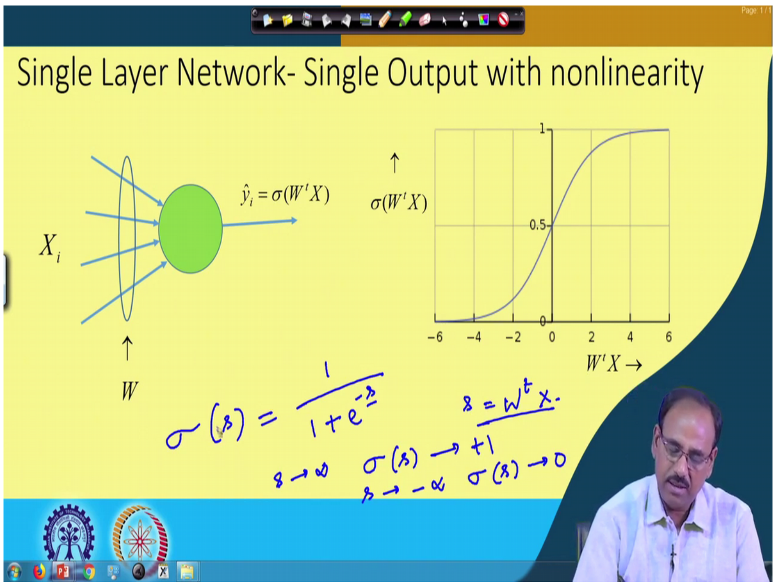
That a kind of nonlinearity that we have used; were simple threshold nonlinearity. That means, if W transpose X is greater than or equal greater than 0; we have put the output to be 1 in the moment it is less than 0; the output was clamped or truncated at 0; that means, the threshold value was set at W transpose X equal to 0. So, the moment the output of the neuron becomes more than 0 it is clamped at 1; if it is less than 0, the output is set to 0.

Now, here you remember that when we talked about the training of the neural network or updating the weights of the neural network, our training procedure makes use of the gradient descent procedure. That means, we have to take the gradient of the error function or the gradient of the loss function and the loss functions or the error functions are computed based on the feature vectors which are misclassified. If the feature vectors, which are correctly classified for them; I do not have to take any action or I do not have to correct or update the weight vectors.

But the feature vectors which are misclassified for them with them I have to define a loss function or have to define an error function. And the weights are updated in such a way that the loss or the error is minimized. And for that, in gradient descent procedure what you do is you take the gradient of the loss function or you take the gradient of the weight function with respect to the weight vector W. And for that if I want to take that gradient it is necessary, that your loss function or the error function should be differentiable, because the gradient is nothing, but our differential operator.

Whereas, if I put the output nonlinearity which is also known as the activation function of the neurons. So, if the activation function of the neurons is a threshold function a threshold function is not differentiable. Because, I have an abrupt change at W transpose X equal to 0; hence the threshold function is not differentiable though it is a very very simple nonlinearity.

(Refer Slide Time: 06:23)



Single Layer Network- Single Output with nonlinearity

$$\sigma(s) = \frac{1}{1+e^{-s}} \qquad s = W^t x.$$

$$s \to \infty \quad \sigma(s) \to +1$$

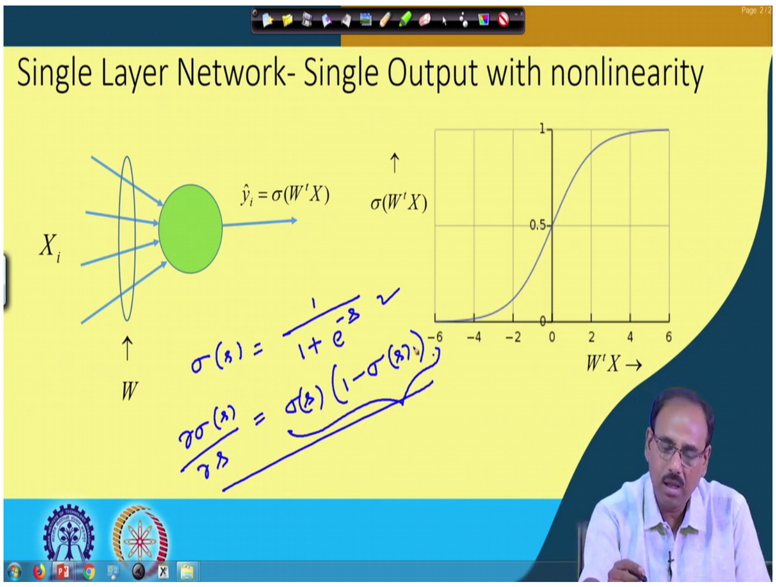
$$s \to -\infty \quad \sigma(s) \to 0$$

So, instead of using the threshold function a short of nonlinearity, which is used is what is the sigmoidal function. We have also talked about the sigmoidal function when we have discussed about the nonlinearity. So, the sigmoidal function is simply given as say I can represent sigmoidal of some function s; some argument s is equal to 1 over 1 plus e to the power minus s.

So, here you find that as s is equal to 0; at s is in our case this s is nothing, but W transpose X, which is the weighted sum of the feature vector components weighted by the corresponding weight component ok. So, s is this. So, at s equal to 0; sigma s is equal to 0.5, which is over here. And as s goes on increasing, say as s tends to infinity; sigma x sigma s tends to plus 1. And as s tends to minus infinity; sigma s tends to be 0; so, that is what is given by this particular curve. And at s equal to 0 sigma s is equal to 0.5 ok.

So, you find that as s increases on the positive side, the sigmoidal function asymptotically reaches value equal to 1. And as s decreases on the negative side, the sigmoidal function asymptotically it reaches value equal to 0. The other advantage is this is a differentiable function and also for when s is sufficiently high to be positive, I can as consider the output to be equal to 1. And if it is sufficiently low that is on the negative side, I can consider the output of the neural network or output of the sigmoidal function to be 0; indicating whether the class is omega 1 or class is omega 2.

(Refer Slide Time: 08:49)



Single Layer Network- Single Output with nonlinearity

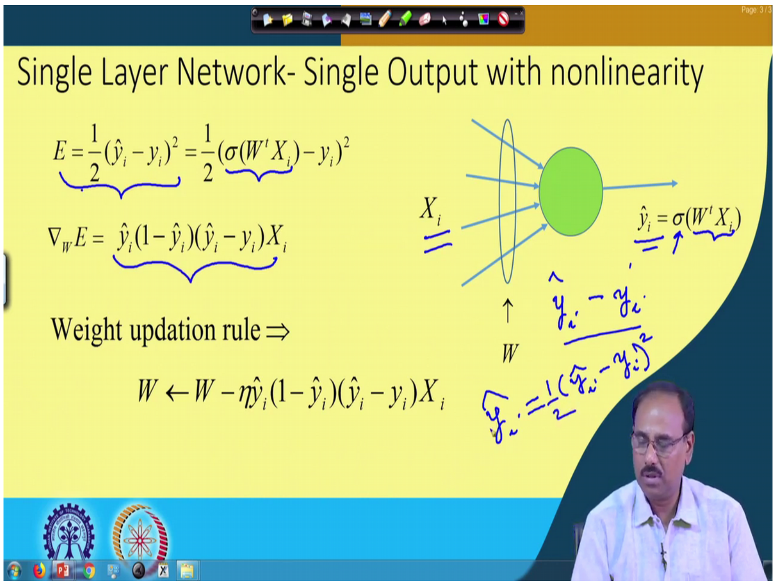
$\sigma(s) = \frac{1}{1+e^{-s}}$

$\frac{\partial \sigma(s)}{\partial s} = \sigma(s)(1-\sigma(s))$

The other advantage that you find that, if I take the derivative of sigma so, I my sigma s was 1 over 1 plus e to the power minus s. So, if I take the derivative of sigma s with respect to s this simply becomes sigma s into 1 minus sigma s, which is also a very very simple form. So, you can take the derivative of this with respect to s and you can verify that actually this is the derivative that you get. So, these are the other advantages or many of the advantages of using sigmoidal function as a non-linearity or as an activation function of the neural network right.

So, given this, so, again I consider a single output, but now this neuron is with a non-linear activation function and the nonlinearity I consider is a sigmoidal function.

(Refer Slide Time: 09:49)



Single Layer Network- Single Output with nonlinearity

$$\underbrace{E = \frac{1}{2}(\hat{y}_i - y_i)^2}_{} = \frac{1}{2}(\sigma(\underbrace{W^t X_i}_{}) - y_i)^2$$

$$\nabla_W E = \underbrace{\hat{y}_i(1-\hat{y}_i)(\hat{y}_i - y_i)X_i}_{}$$

Weight updation rule $\Rightarrow$

$$W \leftarrow W - \eta\hat{y}_i(1-\hat{y}_i)(\hat{y}_i - y_i)X_i$$

X_i
W
$\hat{y}_i = \sigma(W^t X_i)$
$\hat{y}_i - y_i$
$y_i = \frac{1}{2}(\hat{y}_i - y_i)^2$

So, given this, now how can such a single their neural network can be trained. So, as before I assume that my training samples are given as ordered pairs X i, y i; X i being the feature vector and y i is the class level of that feature vector.

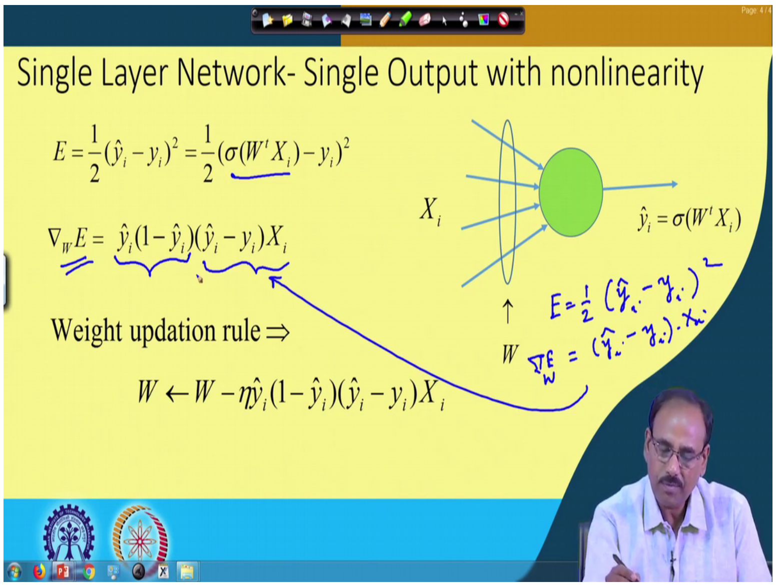
Here, when I feed in this input vector X i to this neural network, I get an output y i hat. In the previous case without nonlinearity this y i hat was simply W transpose X i. But now I impose a nonlinearity by the sigmoidal function so, y i hat is now sigma of W transpose X i right. So, this is my y i hat. Whereas, the class level for this X i is given as y i; so, as a result I have an error which is given by y i hat minus y i. So, this is the error if my output y i hat does not agree with the class level y i that is given.

So, using this error I can define again a loss function or an error function which is nothing but E is equal to half of y i hat minus y i square. So, you find that now the procedure that I am following is a stochastic optimization procedure. If I take the sum of this error over all the samples i equal to 1 to n the kind of optimization procedure that I will be using is a batch optimization procedure. So, if I take single vectors, it is a stochastic optimization procedure.

So, let us now continue with the stochastic optimization. So, my error is given by half of y i hat minus y i square which is nothing but, half of sigmoidal function of W transpose X i minus y i square. So, now, if I take the gradient of this error function or this loss function, the gradient with respect to W; you find that the gradient will be given by this.

You find that in the earlier case, when our y i hat in absence of non-linear absence of nonlinearity was y i hat minus y i square half of this then the gradient of sorry this is not y i hat this is.

(Refer Slide Time: 12:41)



Single Layer Network- Single Output with nonlinearity

$$E = \frac{1}{2}(\hat{y}_i - y_i)^2 = \frac{1}{2}(\sigma(\underline{W^t X_i}) - y_i)^2$$

$$\underline{\underline{\nabla_W E}} = \underbrace{\hat{y}_i(1-\hat{y}_i)}(\underbrace{\hat{y}_i - y_i)X_i}$$

Weight updation rule $\Rightarrow$

$$W \leftarrow W - \eta \hat{y}_i(1-\hat{y}_i)(\hat{y}_i - y_i)X_i$$

$$E = \frac{1}{2}(\hat{y}_i - y_i)^2$$

$$\nabla_W E = (\hat{y}_i - y_i) \cdot X_i$$

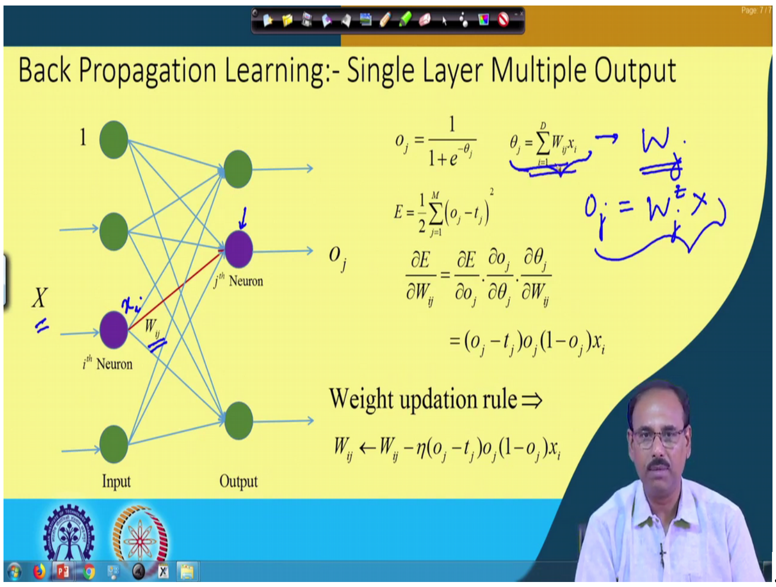
When my error was E is equal to half of y i hat minus y i square; then gradient of E with respect to W was simply, y i hat minus y i times X i this was without nonlinearity. Now, as we have imposed nonlinearity, which is a sigmoidal function and we have said just before that for sigmoidal function if you take the gradient, if you take the derivative with respect to argument it becomes sigmoidal s into 1 minus sigmoidal s.

So, here just before because of that if I take the gradient of E; gradient of the error function the gradient of error function becomes y i hat into 1 minus y i hat where this y i hat is nothing but my sigmoidal which is sigma times W transpose X i. And the other one terms simply comes from here, this is basically gradient of E with respect to W without the sigmoidal function or without the nonlinearity.

So, this is the gradient of E or gradient of the error function, that we get when the sigmoidal nonlinearity is imposed. And once you do that, now my weight updation rule following the gradient descent procedure simply becomes W gets W minus eta again the rate of convergence, convergence factored into y i hat into 1 minus y i hat into y i hat minus y i times X i; this is my weight updation rule. In case I have a single output layer node or a single layer neuron with only one node and by assuming nonlinearity of the neural neurons ok.

And here again you can compare that this is a rule, which is similar to same perceptron algorithm that we considered earlier hence the perceptron network. So, the network that we are talking about right now here is, what is known as a single layer perceptron because, I have only one layer in this neural network. And that too I have only one node in the neural network, to implement a two class problem; if the number of classes are more than 2; then I have to go for more than 2 neurons, but again that number of layers will be 1.

(Refer Slide Time: 15:37)



Back Propagation Learning:- Single Layer Multiple Output

$$o_j = \frac{1}{1+e^{-\theta_j}} \qquad \underbrace{\theta_j = \sum_{i=1}^{D} W_{ij} x_i} \rightarrow W_j^t$$

$$o_j = \underbrace{W_j^t X}$$

$$E = \frac{1}{2}\sum_{j=1}^{M}\left(o_j - t_j\right)^2$$

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial o_j}\cdot\frac{\partial o_j}{\partial \theta_j}\cdot\frac{\partial \theta_j}{\partial W_{ij}}$$

$$= (o_j - t_j)o_j(1-o_j)x_i$$

Weight updation rule ⇒

$$W_{ij} \leftarrow W_{ij} - \eta(o_j - t_j)o_j(1-o_j)x_i$$

So, let us see such a neural network now. So, now, I will consider a neural network again a single layer neural network; but, the number of neurons in the output layer is more than 1. That means, we are considering multiple classes multiclass problem where the number of classes is more than 2.

So, here as we said before that I will have two layers; one of them there is an input layer and this input layer does nothing other than simply passing the input to the output. So, if I take in this i th neuron i th neuron gates x i, which is i th component of the feature vector X and simply passes this to the output ok. So, the output of this neuron is also X i that is what this input neurons does; it simply passes the input to the output nothing else it is just for an interface.

The actual classification is done by this output layer neuron and when I discuss this, I also assume that this output layer neurons have nonlinearity as activation function and as we have done before just in the previous problem. The nonlinearity reconsidered in this case is sigmoidal nonlinearity or a sigmoidal function right.

So, if I consider an i th neuron, the output of the i th neuron in the input layer is x i which is i th component of my feature vector X ok. And this x i is fed to the inputs of all the neurons in the output layer. So, in the output layer, now if I consider a j th neuron so, this i th neuron in the input layer is connected to the j th neuron to the output layer wide a connection weight which is W i j. You find that when we introduced, the feed forward neural network in its totality we had put an index a superscript which was k; two indicating the layers right. So, superscript k indicates that this is a connection from the i th layer from the i th node in k minus first layer to the j th node in k th layer.

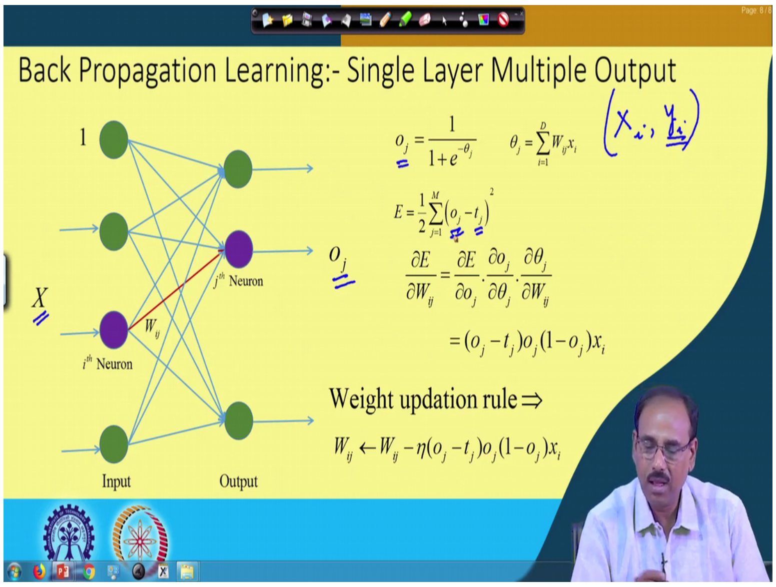
Now, since we are talking about a single layer neuron neural network; so, I will not use that superscript k because, it is simply connection from input layer to the output layer that is known. So, this superscript k I will not use for this purpose. So, let us remove this superscript k. So, I have this situation that x i, which is the output of the i th node in the input layer is connected to the j th node in the output layer through a connection weight x i j.

So, every node in the input layer is connected is feeding input to every node in the output layer of the j th layer. As a result, the weighted sum of all the inputs collected by the j th node in the output layer is given by theta j is equal to W i j times x i; where i varies from 1 to D. Assuming D to be the dimensionality of the feature vectors the input feature vector has got D number of components.

So, this is the sum of or weighted sum of the inputs or this is nothing but if I represent all the weights from the input layer to the j th layer by say W j the weights from all the nodes from the input layer to the j th layer to the j th node in the output layer if those connection weights are represented by a vector W j; this expression theta j is nothing but W j transpose X. So, I can also put in this vector form or this is a scalar form, but both of them are same.

So, that is the weighted sum of the inputs and then we said that every neuron has a sigmoidal activation function. So, the output o j that I am getting from the j th neuron is given by a sigmoidal function of theta j which is nothing but 1 upon 1 plus e to the power minus theta j right.

(Refer Slide Time: 20:15)



Back Propagation Learning:- Single Layer Multiple Output

$$o_j = \frac{1}{1+e^{-\theta_j}} \qquad \theta_j = \sum_{i=1}^{D} W_{ij} x_i \qquad (X_i, y_i)$$

$$E = \frac{1}{2}\sum_{j=1}^{M}\left(o_j - t_j\right)^2$$

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial o_j}\cdot\frac{\partial o_j}{\partial \theta_j}\cdot\frac{\partial \theta_j}{\partial W_{ij}}$$

$$= (o_j - t_j)o_j(1-o_j)x_i$$

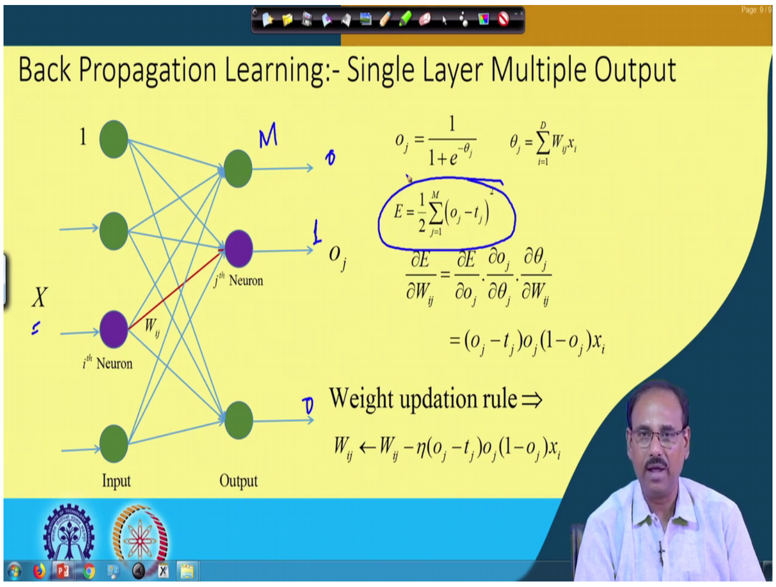
Weight updation rule $\Rightarrow$

$$W_{ij} \leftarrow W_{ij} - \eta(o_j - t_j)o_j(1-o_j)x_i$$

Again we remember that our input vectors are given by X i y i, where y i is the class node. So, if this input X is said that it belongs to class j; that means, we know that what is the output of the j th neuron, which should be the output of the j th neuron when this x is fed to the input of the neural network and that is what we are representing by t j; which is the expected output or the true class of the input vector X which is preferred to the neutral network. But o j is the actual value that you are getting from the j th node of the neural network, when our when you are feeding the same input vector x. So, as a result you have an error which is o j minus t j.

So, as before we define the sum of squared error which is nothing but, o j minus p j square because now output is a vector right for every input X i will have if there are say m number of classes I have j number of M number of nodes in the output layer every output layer node will give me some value.

(Refer Slide Time: 21:31)



Back Propagation Learning:- Single Layer Multiple Output

$o_j = \frac{1}{1+e^{-\theta_j}}$ $\theta_j = \sum_{i=1}^{D} W_{ij}x_i$

$$\boxed{E = \frac{1}{2}\sum_{j=1}^{M}(o_j - t_j)^2}$$

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial o_j}\cdot\frac{\partial o_j}{\partial \theta_j}\cdot\frac{\partial \theta_j}{\partial W_{ij}}$$

$$= (o_j - t_j)o_j(1-o_j)x_i$$

Weight updation rule ⇒

$$W_{ij} \leftarrow W_{ij} - \eta(o_j - t_j)o_j(1-o_j)x_i$$

So, if this input vector X belongs to j th class on the output of the j th node in the output layer should be equal to 1; all other outputs should be 0; so, that is what is my target vector. And using this what is my target vector and what is the actual vector that I get we define what is this sum of squared errors.

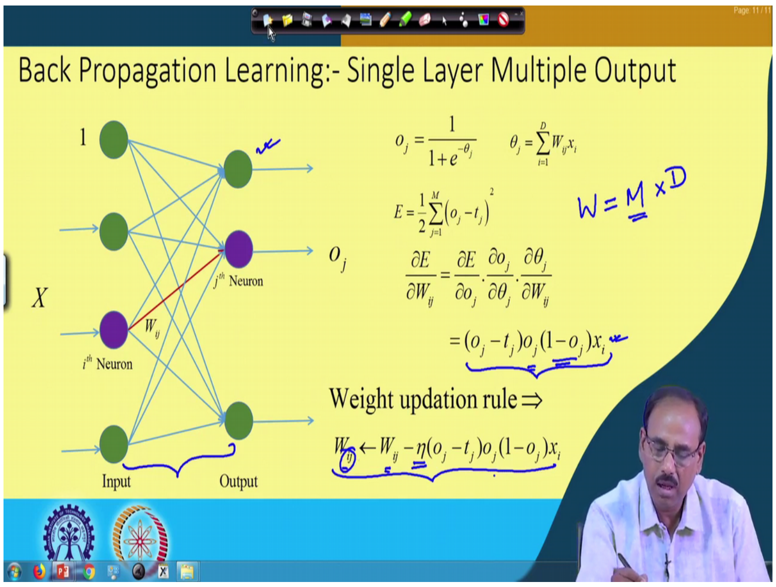
So, again for telling this neural network or for learning I use the back propagation learning algorithm for that what you have to find out is the gradient for using gradient descent approach. So, if I take the gradient of E which is the loss function or the error function with respect to W i j which is the weight component connecting the i th note from the input layer to the j th node in the output layer. So, this gradient del E del W i j I can compute this using chain rule it is del e del o j, where o j is a function of theta j ok, because o j is nothing but sigma theta j.

So, I compute del o j upon del theta j into del theta j again del theta j is a function of W i j; so, del theta j upon del i j del W i j. And through this chain rule, you find that what i get as del E del W i j is nothing but o j minus t j o j is the output of the j th node the actual output of the j th node t j is the target output. So, del E upon del W i j becomes o j minus t j into o j into 1 minus o j times x i where x i is the i th component of the input vector or x i is the output of the i th neuron in the input layer. So, this is the gradient that we will get.

So, using the weight updation rule using the gradient descent procedure the weight updation rule for the weight component W i j as before will be simply W i j is equal to W i j minus some constant, the convergence rate eta into o j minus t j into o j into 1 minus o j times x i ok; this is the weight updation rule. So, if you do it for every i and every j you are updating every component of the weight vectors which are connecting the outputs of the input layer nodes to the inputs of the output layer nodes.

So, once you do this for all i j; so, all the weight components now you have find that we have two sets all multiple sets of weights right; because for every output node I have an weight vector. So, since there are M number of nodes here instead of a single vector I have a matrix.

(Refer Slide Time: 25:07)



Back Propagation Learning:- Single Layer Multiple Output

$$o_j = \frac{1}{1+e^{-\theta_j}} \qquad \theta_j = \sum_{i=1}^{D} W_{ij}x_i$$

$$E = \frac{1}{2}\sum_{j=1}^{M}\left(o_j - t_j\right)^2$$

$$W = \underline{M} \times D$$

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial o_j}\cdot\frac{\partial o_j}{\partial \theta_j}\cdot\frac{\partial \theta_j}{\partial W_{ij}}$$

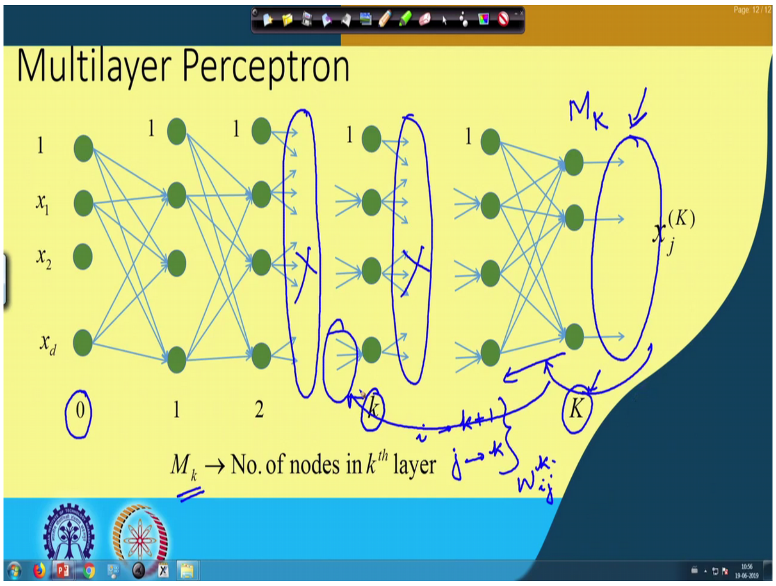
$$= \underbrace{(o_j - t_j)o_j(1-o_j)x_i}$$

Weight updation rule ⇒

$$\underbrace{W_{ij} \leftarrow W_{ij} - \eta(o_j - t_j)o_j(1-o_j)x_i}$$

So, I can put this matrix as W; where W will have say M number of rows and right, D number of columns ok. Where every row corresponds to weight vector of a particular class right. So, later on when we talk about multiple multilayer perceptron with multiple nodes at the output very often we will use this matrix convention other than single vector convention. So, this is the weight updation rule or the training procedure for a two layer network having multiple nodes in the output layer right. So, given this now we can go for training of the feed forward neural network having multiple number of hidden layer nodes and also multiple number of output nodes.

(Refer Slide Time: 26:27)



So, for that the network that we have considered was having this kind of architecture which also we said before that I have an output layer which I put as k th layer the input layer over here. So, this is the output layer which is K th layer I am putting it as capital K there is the input layer and as we said before in the function of the input layer is simply passed the input vector to its output.

So, which we are representing as j as 0 th layer in between a k th layer represented by lowercase k and as we said before that we will also assume that from a node i in k plus first layer to a node j in k th layer I have a connection weight which is given by W i j k. So, this is the convention that I will use. I will also use the convention that in the k th layer the number of nodes is given by M k.

So, M k is the number of nodes in the k th layer. So, while doing this, in the output layer which is capital K; the number of nodes will be given by M capital K, which is same as the number of categories or the number of classes we will consider. Now, unlike in the previous case, here I will have two distinct situations; one is as we said that we can only compute the error at the output because there only at the output I know what is the target. I cannot compute error at any of these layers, I cannot compute error here I cannot compute error here because here I do not know what are the targets. That is the reason these all these layers are known as hidden layers. And this is the layer where the output is visible and the output is known for known classes this is a output layer or visible there all rest of the layers are hidden layers.

So, we will see that when we talk about the back propagation learning of such a multilayer perceptron or a feed forward neural network, then we will have a little difference because now I have a number of hidden layers, which I did not have earlier. So, error that you compute at the output that can directly propagated to the connections between the output layer and the hidden layer just before that. So, here computing the gradient of the error is straightforward. But when we tried to update the weights of the layers in between hidden layers, then I have to see that how this error actually propagates to this level and that is where we will have a little bit of mathematics.

So, we will talk about this training or back propagation learning of the feed forward neural network in our next class.

Thank you.