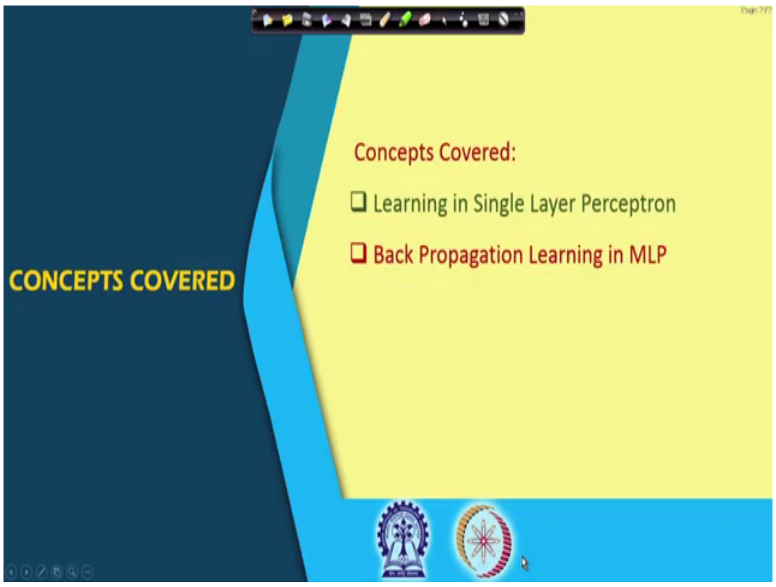
# Deep Learning

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

## Lecture – 23
## Backpropagation Learning

Hello, welcome to the NPTEL online certification course on Deep Learning. We have started our discussion on the neural network particularly the feed forward neural network.

(Refer Slide Time: 00:41)



CONCEPTS COVERED

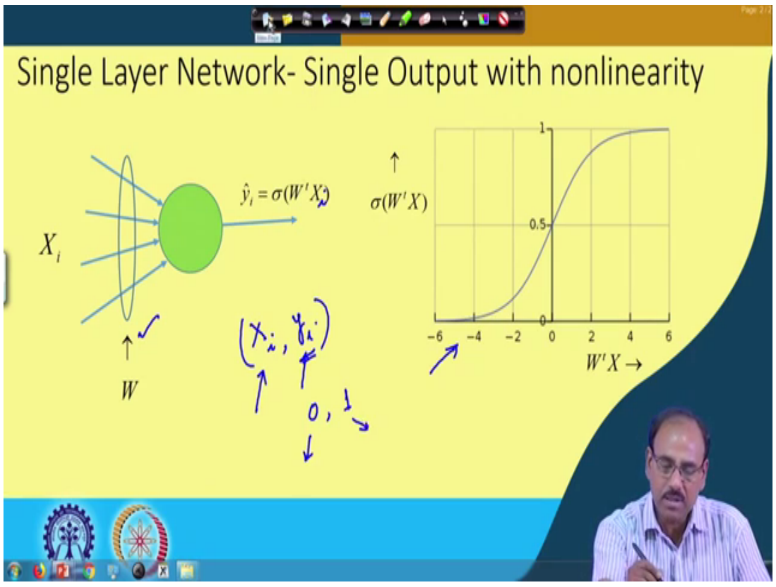
Concepts Covered:

❑ Learning in Single Layer Perceptron

❑ Back Propagation Learning in MLP

And, in the previous class we have discussed about or we have started our discussion on learning in a feed forward neural network. We have discussed about learning in a single layer perceptron, in today's class we will talk about the Backpropagation Learning in multilayer perceptron or multilayer feed forward neural network.

(Refer Slide Time: 01:05)

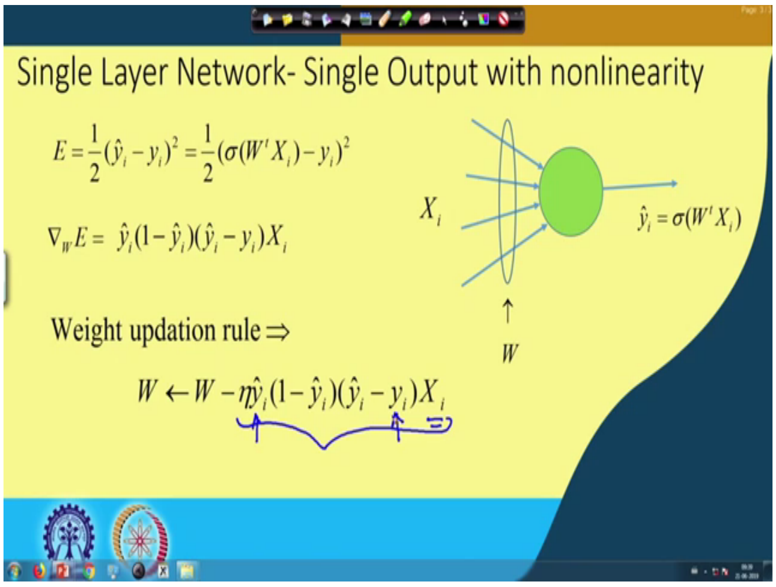


Single Layer Network- Single Output with nonlinearity

So, just to recapitulate what we did in our previous lecture is we had taken single layer neural neurons neural networks with nonlinearity as well as without nonlinearity at the output. So, when we have considered a single layer neural network with nonlinearity at the output. And, the nonlinearity that we have considered was a sigmoidal function as given in the right hand side in the slide. So, this is the sigmoidal function that we have considered and for such a kind of network the output of the neuron which is given as y i hat is equal to sigma of W transpose X. Where W is the weight vector at the input side of the neuron and X i is the input vector.

So, for training you remember that we have said that we obtain the training vectors as ordered pairs given as X i y i, where X i is the feature vector and y i is the class to which this feature vector belongs. And, when I have a single output in the neural network or only one neuron at the output layer actually we are considering a two class problem. And, in to class problem this y i can take value of 0 or 1, 0 means it belongs to one class and 1 means it belongs to another class.

(Refer Slide Time: 02:57)



## Single Layer Network- Single Output with nonlinearity

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

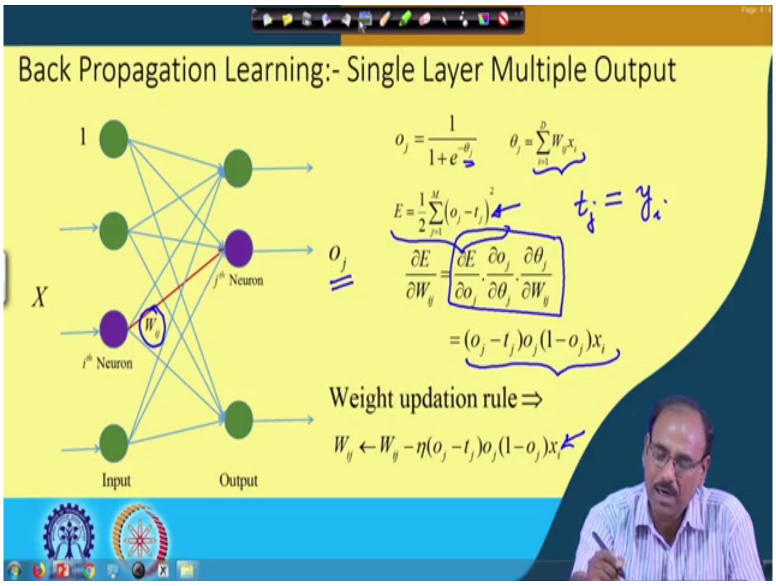
$$\nabla_W E = \hat{y}_i(1-\hat{y}_i)(\hat{y}_i - y_i)X_i$$

Weight updation rule $\Rightarrow$

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

So, given this we have seen that the training algorithm or the weight updation algorithm was obtained as W that is the weight vector is updated as W minus some eta times y i hat into 1 minus y i hat into y i hat minus y i times X i. Where, X i is the input vector and y i hat is the computed output and y i is the desired output; it is the class index of the class to which X i belongs. So, this is what we have obtained with for a single output with nonlinearity where nonlinearity was a sigmoidal function.

(Refer Slide Time: 03:53)



## Back Propagation Learning:- Single Layer Multiple Output

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

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

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

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

Weight updation rule $\Rightarrow$

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

Then we had moved on to single layer network with multiple outputs and when we have multiple outputs then; obviously, I have to consider and weight of the form W i j. So, what I have considered is from the input layer if I take an ith neuron and we have considered that this ith neuron in the input layer is connected to the jth neuron of the output layer through a connection weight which is given by W i j.

So, training in this case means that I have to find out the optimal values of W i j for all values of i and j. So, as you vary the index i; that means, I am considering different neurons at the input layer as i y the index j; that means, I am considering the connection to all neurons in the output layer. And as before if I consider the output of the jth neuron which were put in here as O j, O j is a sigmoidal function of theta j.

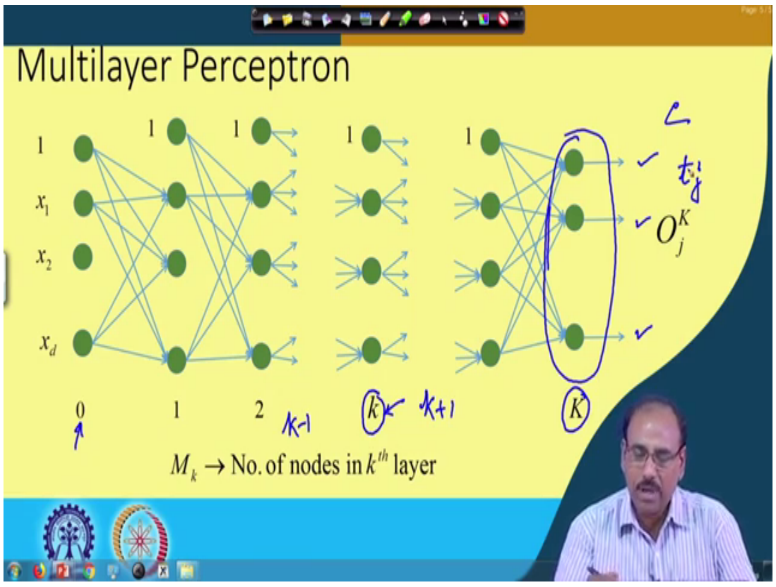
Where theta j is nothing but weighted sum of all the input feature components or in other words this is the dot product of W i j for all values of i the vector that I get with the input vector x. And given this we had defined an error function or a loss function which is half of o j minus t j square where t j is the target output or I can say this t j is nothing but y j.

So, the t j that you have considered it is nothing but y j that is the actual class belongingness of the feature vector. So, what we need to do is, we need to minimize this error function with respect to W i j or this loss function with respect to W i j. So, for that again we are using the gradient descent approach, so I have to find out what is the gradient of this loss function with respect to the weights.

So, I compute del E del W i j which according to chain rule comes out to be del E del o j into del O j del theta j into del theta j del W i j. And, if you compute this comes as o j minus t j into o j into 1 minus o j times x i. And, given this my weight updation rule in this case becomes W i j gets W i j minus theta times o j minus t j into o j into 1 minus o j times x i; where x i is the ith component of the input vector.

And, W i j is the connection weight from the ith node in the input layer to the jth node in the output layer. So, this is the updation rule that we get with in a single layer perceptron when I have number of outputs or multiple number of outputs.

(Refer Slide Time: 07:22)


## Multilayer Perceptron

$M_k \rightarrow$ No. of nodes in $k^{th}$ layer

Now, let us go to a multilayer perceptron. So, as we just indicated in the previous class that when I consider a multilayer perceptron or multilayer feed forward network then; obviously, I have to have an input layer which receives the input vector. And, I have to have the output layer which tells you the class if or class belongingness of the input vectors. So, if I have say c number of classes then I will have c number of neurons at the output layer, so accordingly I will have c number of outputs.

And if a feature vector belongs to say class 5, then output of the 5th neuron should be high and the output of all the neurons should we low. In this case in this particular example we are assuming that there are K number of layers where capital K is the output layer at the input layer we are considering that this is 0th. And, we said in the previous class that the neurons in the 0th layer simply passes the input to it is output, it does not perform any other function.

Whereas in all the hidden layers the neurons perform two tasks, every neuron takes the weighted sum of all the inputs that it receives from it is previous layer. And then computes a non linear function over it and the nonlinearity that we are considering over here is nothing but a sigmoidal nonlinearity. And, if I consider any layer say kth layer where this k I represent as a lowercase in lowercase. So, every kth layer passes it is output to k plus first layer and it receives inputs from k minus first layer. So, every node in the k minus first layer passes output to every node in the kth layer, and every node in the kth layer passes the output to every node in the k plus first layer.

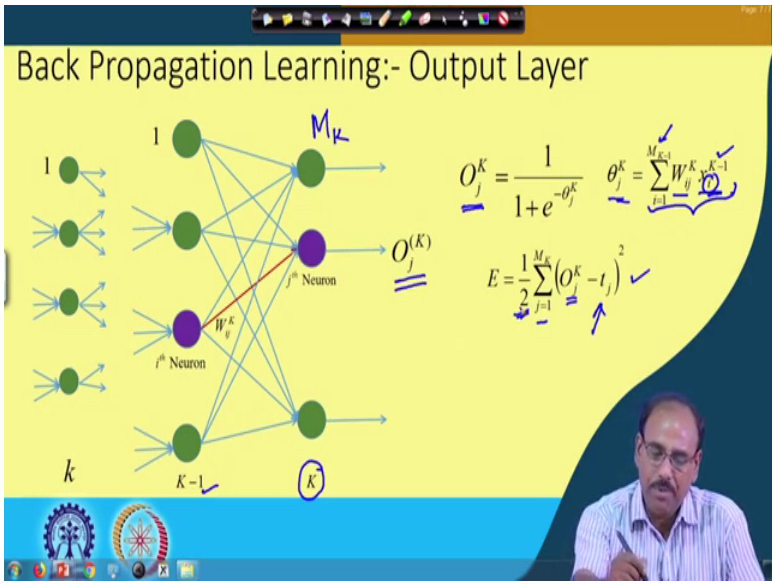
Now, coming to the error function of the loss function which we want to optimize while training this neural network, you can easily emerge in that we can only compute the error function of the loss function at the output layer. Because it is only at the output layer I know what is my target output t j. The reason being if i say that a feature vector or training vector belongs to jth class I know that t j should be high or ideally t j should be 1, And all other outputs except t j or except the output of the jth neuron in the output layer should be low or ideally they should be 0.

So, this is known and as the expected output or the ground truth at the output layer is known I can only compared the error function at the output layer. I cannot compute error function in any of the layers any other layers, the reason being I do not know what is the expected output or what is the target output at the outputs of any other layers. And that is the reason all other layers except the output layer and the input layer of course, are known as the hidden layers because I do not know what is their outputs.

So, given this now let us see that how we can train this neural network. So, over here the training unlike in case of a single layer neural network where training was very simple, because I had to update the connection weights from the input layer to the output layer. Now, in this case the training for the output layer and training for the hidden layers will be slightly different. Because, I can compute the error at the output layers, and once I compute the error at the output layers I can back propagate that error through the gradient descent approach to the connection weights. In between this K minus first layer K capital; that means, it is the layer just before the output layer.

So, I can back propagate the error for updation of the weight vectors from K minus first layer to kth layer. But when I want to update any of the weight vectors in between the hidden layers you find that I do not have I cannot compute directly what will be the error at the output of any of the hidden layers. So, following chain rule I have to get the feedback or I have to back propagate the effect of the output error; error that you can compute at the output layer to the hidden layers. And that you have to use for updation of the weight vectors in between the hidden layers following or gradient descent approach. So, let us see how we can do it.

(Refer Slide Time: 12:51)



## Back Propagation Learning:- Output Layer

$$O_j^K = \frac{1}{1+e^{-\theta_j^K}} \qquad \theta_j^K = \underbrace{\sum_{i=1}^{M_{K-1}} W_{ij}^K x_i^{K-1}}$$

$$E = \frac{1}{2}\sum_{j=1}^{M_K}\left(O_j^K - t_j\right)^2$$

So, given this you find that updation of the weight vectors at the output layer is almost similar to updation of the weight vectors in a single layer single layer neural network where we had multiple number of outputs. So, here again I assume that this capital K indicates index of the output layer. I take jth neuron in the output layer and the output of the jth neuron is represented by O j K. And, from our previous discussion you can recollect that O j K is a sigmoidal function, it is the sigmoidal function of weighted inputs of weighted sum of the inputs which are coming to the jth neuron.

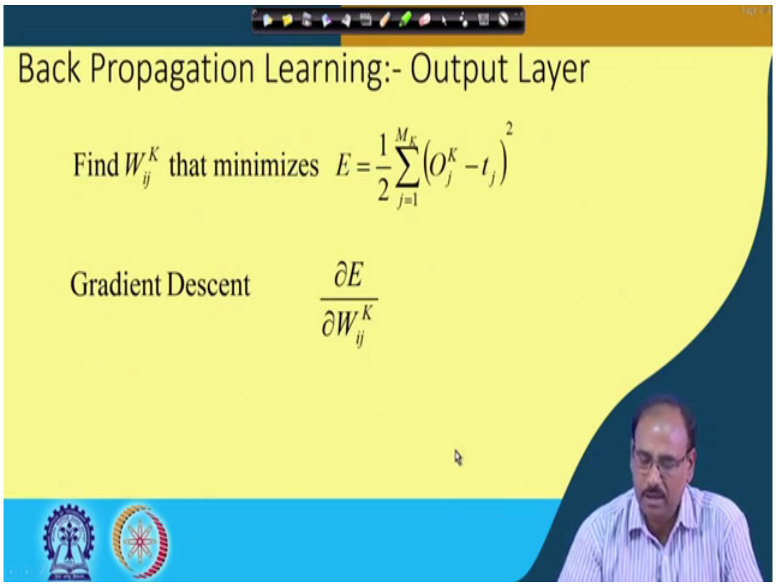
So, that weighted sum of the inputs I am representing as W i j K x i K minus 1, what is this x i K minus 1? This K minus 1 indicates that this is the output of a neuron from the previous layer that is K minus first layer. And this subscript i indicates that it is the output of the ith neuron in the K minus first layer, so, I take this ith neuron in the K minus first layer. This ith neuron is connected to the jth neuron in the kth layer through a connection weight W i j K.

So, the weighted sum of all the inputs to the jth neuron in the kth layer is given by W i j K into x i K, I have to take the sum over i is equal to 1 to M K minus 1. Where M K minus 1 is the number of neurons in the K plus first layer, so this is the weighted sum; and once I have this weighted sum then I have to compute the sigmoidal function over it. So, the sigmoidal function is given by 1 over 1 plus E to the power minus theta j K where this theta j K is the weighted sum. And that is the output of the jth node in the output layer which is O j k.

So, once I have this output I know because all the vectors that we are feeding to the input of this neural network or the training vectors. So, I know what is my target output at the jth node, because if the sample belongs to plus j ideally t j should be equal to 1. And this O j K is the output as computed by the neural network at a particular instant of time known as the folk that is the different steps of training. So, I compute the sum of squared error which is given by O j K minus t j square take the summation over all j for j is equal to 1 to M K, M K is nothing but the number of nodes at the output layer.

So, that gives you the sum of squared errors we take half of this, because as we have seen in our previous class also that we have to take the gradient descent and this is a squared error. So, when I take the derivative this when I take the derivative 2 comes over here, so, that 1 and half that gets cancelled. So, that is the only reason, that were putting a scale factor which is half. So, far updation of the weight vector W i j what I need to do is I have to take the gradient of the derivative of this error E with respect to weight vector W i j K.

(Refer Slide Time: 16:52)



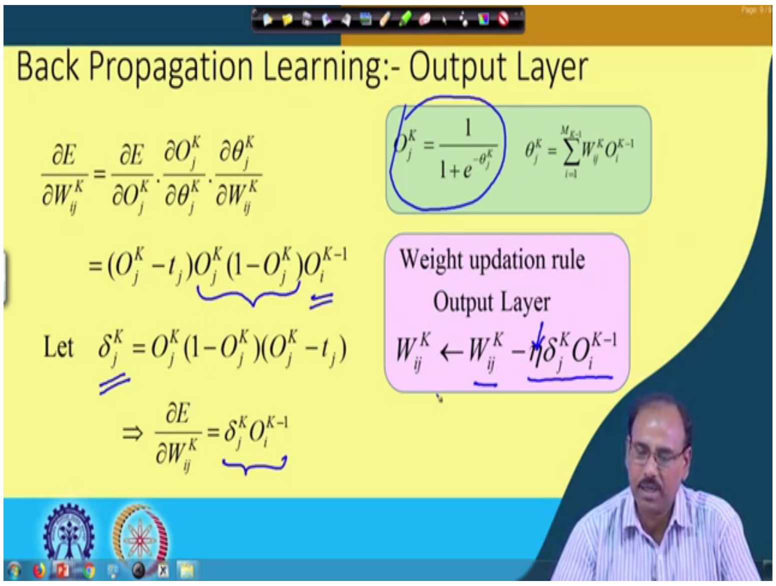
Back Propagation Learning:- Output Layer

Find $W_{ij}^K$ that minimizes $E=\frac{1}{2}\sum_{j=1}^{M_K}\left(O_j^K-t_j\right)^2$

Gradient Descent $\frac{\partial E}{\partial W_{ij}^K}$

So, as you take the derivative of the error or squared error with W i j k, so this is what i M just said I need to take the derivative del E del W i j K.

(Refer Slide Time: 17:02)



## Back Propagation Learning:- Output Layer

$$\frac{\partial E}{\partial W_{ij}^K} = \frac{\partial E}{\partial O_j^K} \cdot \frac{\partial O_j^K}{\partial \theta_j^K} \cdot \frac{\partial \theta_j^K}{\partial W_{ij}^K}$$

$$= (O_j^K - t_j)\underbrace{O_j^K(1-O_j^K)}O_i^{K-1}$$

Let $\delta_j^K = O_j^K(1-O_j^K)(O_j^K - t_j)$

$$\Rightarrow \frac{\partial E}{\partial W_{ij}^K} = \underbrace{\delta_j^K O_i^{K-1}}$$

$$O_j^K = \frac{1}{1+e^{-\theta_j^K}} \qquad \theta_j^K = \sum_{i=1}^{M_{K-1}} W_{ij}^K O_i^{K-1}$$

Weight updation rule

Output Layer

$$W_{ij}^K \leftarrow W_{ij}^K - \eta \delta_j^K O_i^{K-1}$$

And, if I compute this again following the chain rule I get del E del W i j K is equal to del E del O j K. The reason we are using this chain rule is that E is a given as a function of O j that is the output, W i j K comes indirectly. So, we compute this derivative using the chain rule that is del E del O j K times del O j K del theta j k, so if you remember that O j K is nothing but sigmoidal function of theta j K. So, it is del O j K times del theta j K and theta j K is the weighted input or weighted sum of the inputs at the jth node. So, that is given by W i j K times theta i K minus 1.

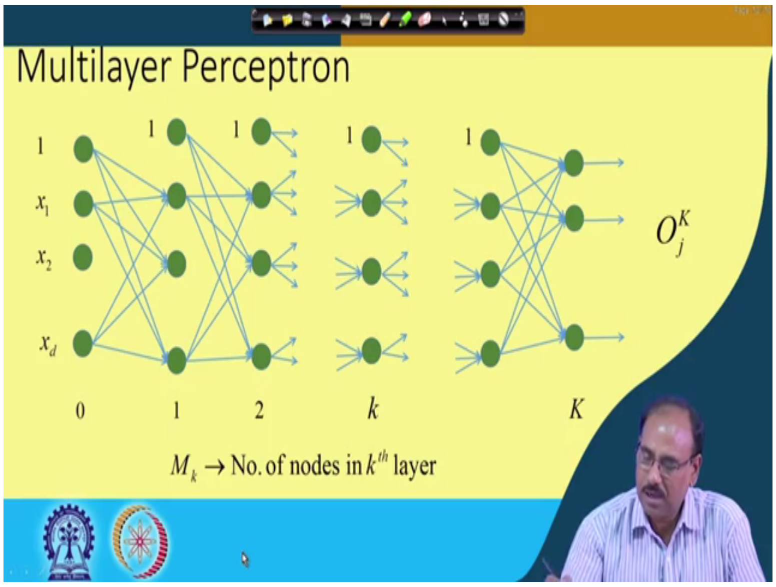
So, the last component in this chain becomes del theta j K del W i j K, and if you compute this you find that del E del theta j K we said that one advantage yeah. So, del E del O j K that simply becomes O j K minus t j because E was half of O j K minus t j square. So, del E del O j K become O j K minus t j, then del O j K del theta j K O j K is a sigmoidal function of theta j K, so as given by this expression. So, del O j K del theta j K becomes O j K into 1 minus O j K we said that the advantage of using sigmoidal function is the derivative becomes very simple which is of this form.

And del theta j K upon del theta j K del W i j K simply becomes O i K minus 1, because if I take the derivative of this with respect to del W i j K it is simply becomes O i K minus 1. So, if I put this O j K into minus 1 minus O j K into O j K minus t j as delta j K the reason why I am putting this as delta j K will become clear when we go for updating of the hidden layer weights. So, if I put this then del E del W i j K simply becomes delta j

K into O i K minus 1, where O i K minus 1 is the output of the ith node in the K minus first layer.

And given this the weight updation rule W i j K simply becomes W i j K minus eta times delta j K O i K minus 1; where this eta as we said before is nothing, but a constant which controls the rate of convergence or the learning rate. So, this is how I can update the weights of the output layer that is the layer between the output layer and the layer just before the output layer, now let us say how we can update the weights of the hidden layer.

(Refer Slide Time: 20:41)



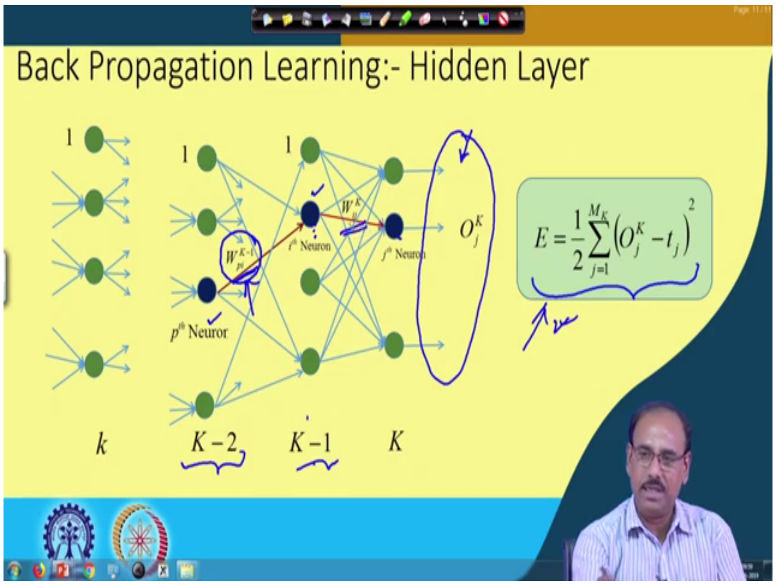
## Multilayer Perceptron

$M_k \rightarrow$ No. of nodes in $k^{th}$ layer

As we said that I can only compute error at the output layer this is where I can compute the error. But when I am updating the weights at the hidden layer I cannot compute what is the error at the output of any of the hidden layers because I do not know what is the target output over here.

So, whatever error I compute here that through back propagation has to be brought to this particular layer and in this layer I have to see what is the effect of this error. And, using that I have to go for weight updation again following the gradient descent approach, so let us see how we can do that.

(Refer Slide Time: 21:24)


Back Propagation Learning:- Hidden Layer
$E = \frac{1}{2}\sum_{j=1}^{M_K}\left(O_j^K - t_j\right)^2$
$O_j^K$
$W_{pi}^{K-1}$
$W_{ij}^K$
$p^{th}$ Neuron
$i^{th}$ Neuron
$j^{th}$ Neuron
1
1
1
$k$
$K-2$
$K-1$
$K$

So, for doing this I assume that initially we have updated the weights between the output layer and the layer just before the output layer, so which was my K minus first layer. Now, K minus second layer is one of the hidden layers, so let us see that how we can update the weights between K minus first layer and K minus second layer. Then we can generalize that result that we get the weight updation rule to any of the hidden layers.

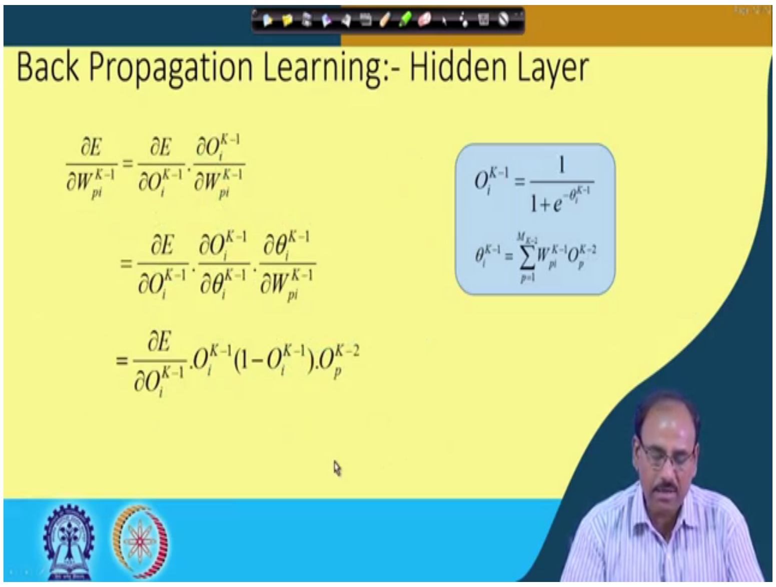
So, in order to do this what I do is I take a neuron say pth neuron in K minus second layer, I take ith neuron in K minus first layer. You follow you see over here that I am just following a chain because in the previous case I am, I have assumed that from ith neuron in the K minus first layer. I have a connection from the jth neuron in the kth layer and that connection weight was W i j K, now I am going one more layer before K minus first layer.

So, in K minus second layer I have this pth neuron and this pth neuron in the K minus second layer is connected to ith neuron in the K minus first layer and ith neuron in the K minus first layer is connected to jth neuron in the kth layer. And I am also assuming, but this pth neuron in K minus second layer is connected to ith neuron in the K minus first layer through a connection weight which is given by W p i K minus 1.

But you remember as we just said that I can compute the error only at the output layer, I cannot compute the error in any of the hidden layers. So, my error function of the loss function is still given by E is equal to half O j K minus t j square j varying from 1 to M K that is output that I am computing the area that I am computing at the output of the output layer, so this is still my error function.

So, now in order to update this connection weight W p i K minus 1 what I have to do is I have to take the derivative of this error function E this loss function E with respect to W p i K minus 1, it is no more with respect to W i j K. So, again I have to follow the chain rule in order to find out that how this error E which is computed at the output layer varies with the variation of W p i K minus one. So, let us apply this chain rule again over here I will just skip this slide.

(Refer Slide Time: 24:42)



## Back Propagation Learning:- Hidden Layer

$$\frac{\partial E}{\partial W_{pi}^{K-1}} = \frac{\partial E}{\partial O_i^{K-1}} \cdot \frac{\partial O_i^{K-1}}{\partial W_{pi}^{K-1}}$$

$$= \frac{\partial E}{\partial O_i^{K-1}} \cdot \frac{\partial O_i^{K-1}}{\partial \theta_i^{K-1}} \cdot \frac{\partial \theta_i^{K-1}}{\partial W_{pi}^{K-1}}$$

$$= \frac{\partial E}{\partial O_i^{K-1}} . O_i^{K-1}(1-O_i^{K-1}).O_p^{K-2}$$

$$O_i^{K-1} = \frac{1}{1+e^{-\theta_i^{K-1}}}$$

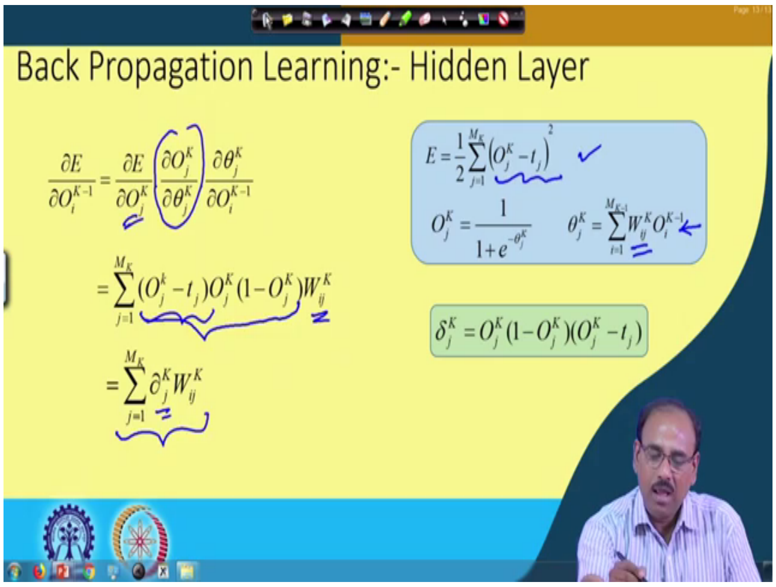
$$\theta_i^{K-1} = \sum_{p=1}^{M_{K-2}} W_{pi}^{K-1} O_p^{K-2}$$

So, this is what I have to do and the derivative that I just said that I have to compute del E del p i K minus 1. So, this is the derivative that I have to take. And following the chain rule you find that I can write this as del E del O i K minus 1 into del O i K minus 1 del W p i K minus 1. And again del O i K minus 1 W p i K minus 1 using the chain rule can be written as del O i K minus 1 del theta i K minus 1 del theta i K minus 1 del p i K minus 1.

So, these two are very simple, because I know that O i K minus 1 is just the sigmoidal function of theta i K theta i K minus 1. And I also know that theta i K minus 1 is nothing but W p i K minus 1 O p K minus 2. And you take the summation from p is equal to 1 to M K minus 2, you remember that we this we are considering K minus second layer. So, number of nodes in the K minus second layer is K minus 2.

So, as these two functions are known I can easily compute what is del O i K minus 1 del theta i K minus 1 which is nothing but this the derivative of the sigmoidal function. And, del theta i K minus 1 del p i K minus 1 which is simply O p K minus 2 that is this. So, what I am left with is del E del O i K minus 1, so how I can compute this del E del O i K minus 1.

(Refer Slide Time: 26:47)



## Back Propagation Learning:- Hidden Layer

$$\frac{\partial E}{\partial O_i^{K-1}} = \frac{\partial E}{\partial O_j^K}\left(\frac{\partial O_j^K}{\partial \theta_j^K}\right)\frac{\partial \theta_j^K}{\partial O_i^{K-1}}$$

$$= \sum_{j=1}^{M_K} \underbrace{(O_j^k - t_j) O_j^K (1 - O_j^K)}\, W_{ij}^K$$

$$= \underbrace{\sum_{j=1}^{M_K} \partial_j^K W_{ij}^K}$$

$$E = \frac{1}{2}\sum_{j=1}^{M_K}\left(O_j^K - t_j\right)^2 \checkmark$$

$$O_j^K = \frac{1}{1+e^{-\theta_j^K}} \qquad \theta_j^K = \sum_{i=1}^{M_{K-1}} W_{ij}^K O_i^{K-1} \leftarrow$$

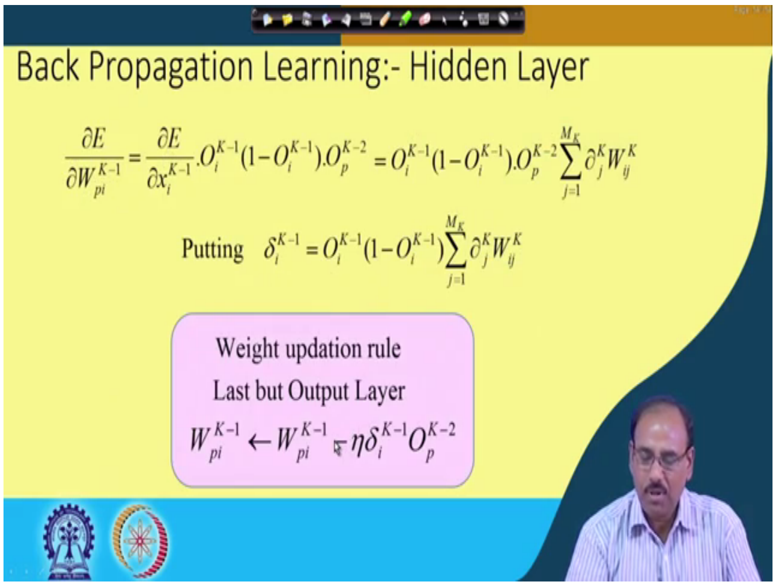
$$\boxed{\delta_j^K = O_j^K (1 - O_j^K)(O_j^K - t_j)}$$

So, over here you find that E is given as we have already said the error at the output layer. And, where O j K is the sigmoidal function of theta j K and theta j K is nothing but weighted sum of O i K minus 1. So, given this you find that del E del O i K minus 1 again by chain rule I can write this as del E del O j K into del O j K del theta j K into del theta j K del O i K minus 1 right.

And that simply becomes sum of O j K minus t j into O j K into O j K minus O j K which is the derivative of this into del theta j K del O K minus 1 is nothing but W i j. So, this is W i j K and del E del O j K is nothing but O j K minus t j it is O j K minus t j, and that has to be summed over all j, so it is again varying from j is equal to 1 to M K.

And, this I can write as now you can try to recall it early we had return this term O j K minus t j into O j K into 1 minus O j K as delta j K. So, that I put over here, so that simply tells me that O del O E del sorry del E del O i K minus 1 simply becomes some of delta j K W i j K. Where you take the summation over j is equal to 1 to M K that is the total number of nodes or the neurons that you have at the output layer.

(Refer Slide Time: 29:00)



## Back Propagation Learning:- Hidden Layer

$$\frac{\partial E}{\partial W_{pi}^{K-1}} = \frac{\partial E}{\partial x_i^{K-1}} . O_i^{K-1}(1-O_i^{K-1}).O_p^{K-2} = O_i^{K-1}(1-O_i^{K-1}).O_p^{K-2}\sum_{j=1}^{M_K}\partial_j^K W_{ij}^K$$

Putting $\delta_i^{K-1} = O_i^{K-1}(1-O_i^{K-1})\sum_{j=1}^{M_K}\partial_j^K W_{ij}^K$

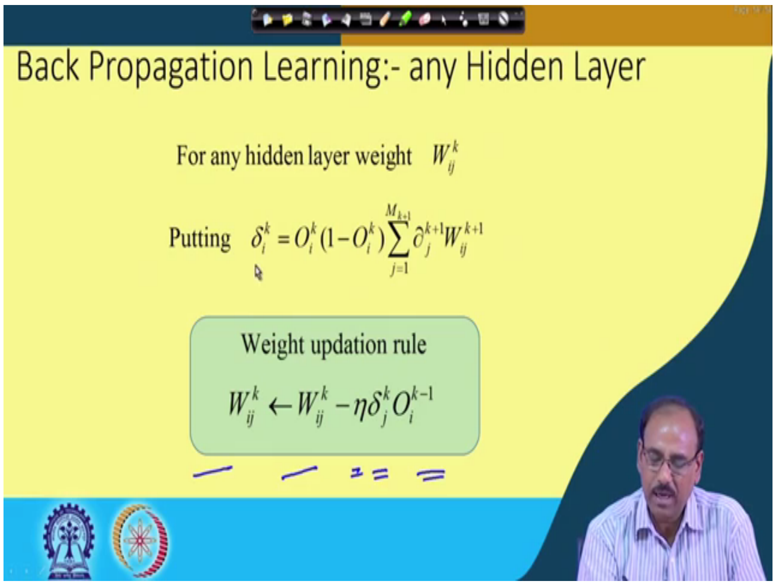
Weight updation rule

Last but Output Layer

$$W_{pi}^{K-1} \leftarrow W_{pi}^{K-1} - \eta\delta_i^{K-1}O_p^{K-2}$$

So, given this now our weight updation rule of the weights between K minus second layer, 2 K minus first layer simply becomes W K minus 1 W p i K minus 1 that is updated as W p i K minus 1 minus eta times del i K minus 1 into sorry. This was the weight updation rule at the last, but output layer, so that was eta times delta i K minus 1 into O p K minus 2. So, this was the weight updation rule between the output layer and the layer before output layer, so in the hidden layer I will have a summation term.

(Refer Slide Time: 29:54)



## Back Propagation Learning:- any Hidden Layer

For any hidden layer weight $W_{ij}^k$

Putting $\delta_i^k = O_i^k(1-O_i^k)\sum_{j=1}^{M_{k+1}}\partial_j^{k+1} W_{ij}^{k+1}$

Weight updation rule

$$W_{ij}^k \leftarrow W_{ij}^k - \eta\delta_j^k O_i^{k-1}$$

Here I am putting delta i k as summation of this as I said that I can generalize this from output layer to any of the hidden layers. So, I am taking any of the hidden there is kth layer at which I can put this delta i k k is lowercase. Means it is any of the hidden layers that simply becomes O i k into 1 minus O i k into summation of the contribution error of error term that you are getting from k minus first layer.

So, that gives you delta i k and using this my weight updation rule in the kth layer that simply becomes W i j k getting W i j k minus eta times delta j k times O i k minus 1. So, this O i k minus 1 is the output of the k minus first layer the ith node of the k minus first layer, and using this I can update the weights of any of the hidden layer.

So, what I have to do is I have to compute this for all values of i j and k and while you do that you have to start from your output layer and then gradually move to all the hidden layers. So, this is what is the weight updation rule of the back propagation learning algorithm in multilayer neural network.

And, you find that here the error function that we have considered is the sum of squared error or that is also known as quadratic error. So, in your in our next class we will try to see that what is the problem that we face with the quadratic error quadratic error and what sort of remedy we can have.

Thank you.