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

Lecture – 25
Backpropagation Learning – Example

Hello, welcome to the NPTEL online certification course on Deep Learning. In previous few classes, we have discussed about the back propagation learning; we have also talked about the different types of loss functions which the back propagation learning algorithm tries to minimize while training a neural network.

Now, as we said earlier that back propagation learning is the heart of the deep learning process. So, understanding the back propagation is very, very important. So, though we have talked about the back propagation learning algorithms before, I want to explore further on back propagation learning with examples, so that the process of back propagation learning is very, very clear to you.

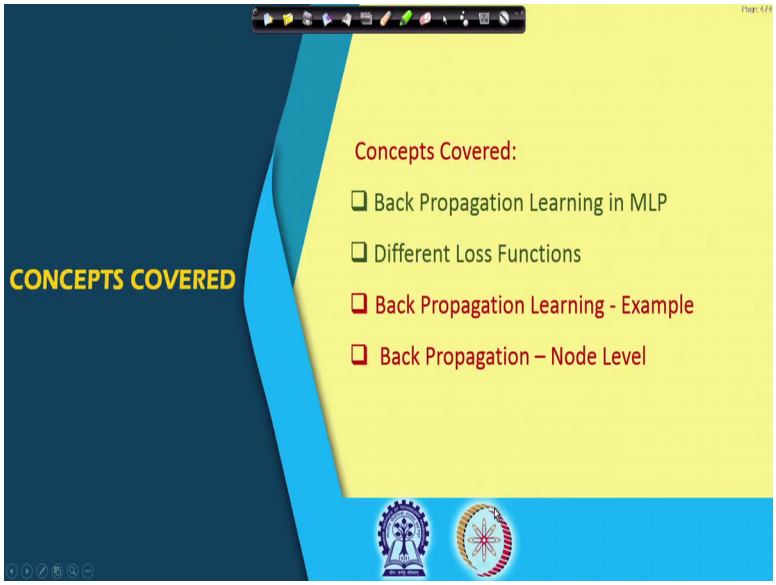
The reason being in all subsequent lectures whenever learning is involved, I will simply refer to as back propagation learning without going into details of the algorithms. So, it is very, very important that you fully understand what is back propagation learning, and what is the mechanism for back propagation learning or even when you write an algorithm, when you write a program involving the learning of a neural network, how you can write simple programs even exploring the parallelism of the machines.

And that is also important because when we talk about deep learning in today's scenario, you will find that the concepts that we discussed in deep learning, those concepts are not very new the concepts even existed long back the neural network is few decades old. But why this branch of machine learning was not so popular earlier or not being used in a broader sense is that, we did not have the computational power.

Now, we have the computational powers in the form of high performance computers, in the form of GPU or graphics processing units, which are massively parallel. So, making use of those parallel processing capability, now implementing the machine learning algorithms working on huge amount of data has become feasible.

So, over there exploiting parallelism is also very very important in order to make your machine learning algorithm successful. So, over there the deep learning or understanding of the deep learning and how the parallelism in deep learning algorithms can be exploited knowing that is very very important for application of deep learning algorithms or development of any deep learning solution for any kind of problem.

(Refer Slide Time: 03:37)



So, what we have done as I said in our previous classes is that we have talked about a back propagation learning in multilayer perceptron. We also talked about the loss functions like sum of squared error loss function, and also we talked about the cross entropy loss function. And over there we have seen that the problem or the drawback of squared error loss function, which sometimes leads to very slow learning rate, particularly when your actual output that you are getting from the network is far away from the target output.

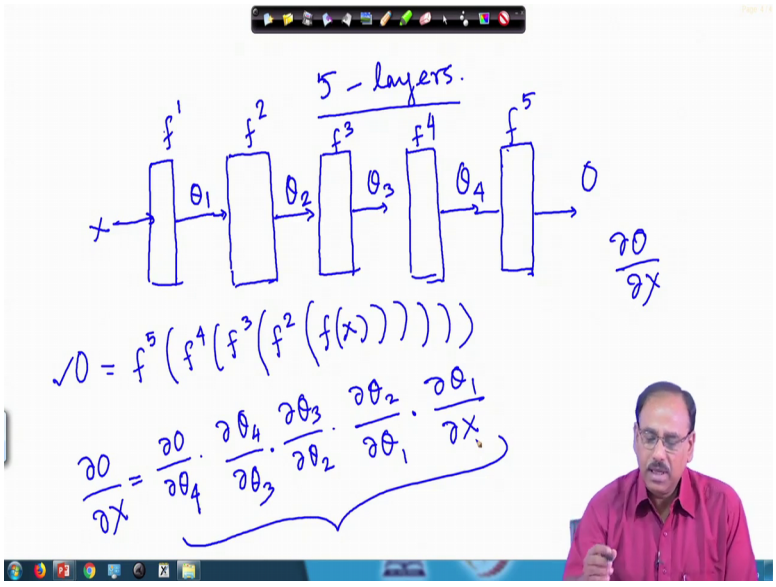
There the derivative almost becomes 0, the derivative almost vanishes, so as a result the learning rate becomes very very low, which in case of cross entropy loss function that problem is avoided because there your learning rate as we have seen before is proportional to the difference between the target output and the actual put that you get.

So, as a result if I use cross entropy loss as loss function to be minimized, the learning rate is much higher than the learning rate that you get in case of using sigmoidal function and the squared error loss. So, today we will further explore on back propagation

learning with some examples. Examples at the network level and also examples at the node level, because within every node which computes a complex function sometimes, we can even explore the smaller functionalities of the back propagation algorithm.

So, let us try to see that how this back propagation algorithm actually works. So, before that I will just discuss about which we have used in case of back propagation learning that is the chain rule of partial derivatives. So, what we have is say I have a feed forward network.

(Refer Slide Time: 06:03)



Let us assume that I have a feed forward network have in 5-layers. So, I have first layer, where the input is actually input vector X. The first layer transforms this x with a mapping function say f 1, and this gives me an output of say theta 1 which goes to next layer of the speed forward network that computes a function f 2 on the input which is fed to it. So, it computes f 2 on theta 2.

And this gives me an output say it computes f 2 on theta 1 and that gives me an output say theta 2 which in turn is fed to the next layer which has a mapping function f 3 and this gives an output of theta 3 and so on, it theta 3 enters f 4 giving you an output of theta 4. And finally, this theta 4 enters the final layer which computes the mapping f 5 and so you get your final output say O right.

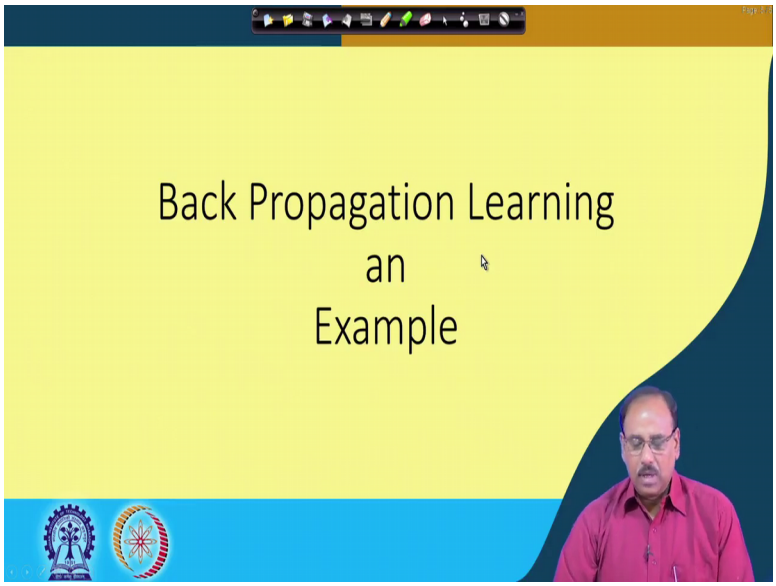
So, now if I want to find out that derivative of O with respect to X, so remember what this what does this derivative actually tell you that derivative tells you that considering o as a function of x, if I make a little perturbation, if I perturb x slightly what is its effect on the final output o and that is what is given by del O del X. Now, in this case, your final output is f 5 ok, so I can say that my final output O which is equal to function f 5 that works on f 4 that works on f 3 which works on f 2, and finally, f 2 works on f of x. So, this is my final output function O.

So, if i want to compute del O del X, I cannot really directly compute del O del X because output O is quite far away from X. So, from input X, I cannot directly see what is O. So, to come to O on the output, I have to pass through the functions f 1, f 2, f 3, and f 4. So, to compute this f X what I have to do is O is directly visible to theta 4 right. So, what I have to compute is, I have to compute del O del theta 4. Then theta 4 which is the output of f 4 that is directly visible to theta 3.

So, I have to compute into del theta 3 del and theta 3 is directly visible to theta 2 sorry theta 4 is directly visible to theta 3. So, I have to compute del theta 4 del theta 3, where theta 3 is visible to theta 2. So, I have to compute del theta 3, del theta 2, then del theta 2, del theta 1 and then finally, del theta 1 and del X.

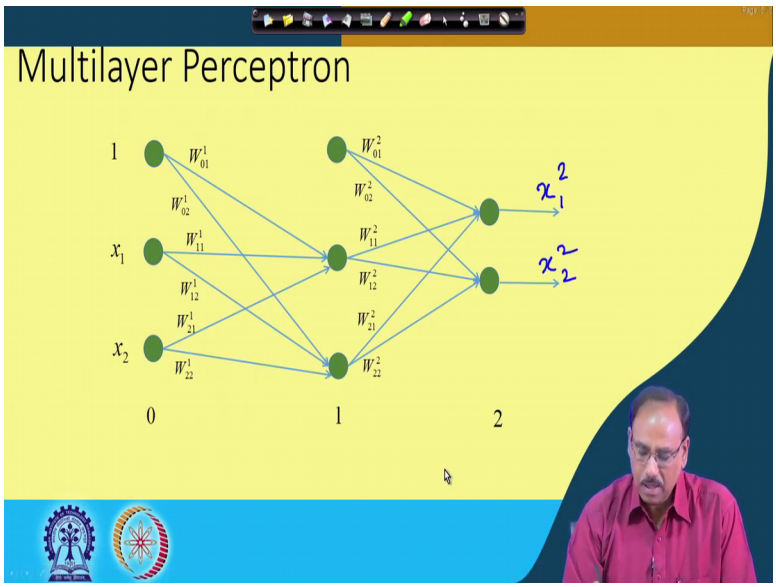
So, you find that this chain of partial derivatives when multiplied together that actually gives you del O del X. And del O del X as we said is nothing but the sensitivity of the output on the input X that is if I X if I change X or perturbed X lightly what is its effect on the final output that is what is given by this partial derivative. And as we have seen here that, when I have a sequence of functions from input to output, I cannot directly compute this partial derivative rather I have to make use of the chain rule of differentiation to find out the partial derivative of output with respect to input. So, in the back propagation learning algorithms, we have made use of these properties of the partial derivatives. So, let us see that what we have done.

(Refer Slide Time: 11:09)



So, now we will explain this back propagation learning algorithm with an example and the example that I am going to take is on the network layer that is considering the neural network as a whole. Though we have explained this algorithm before we have had all the derivations, but I want to explain further and physically see practically see how does it work with the help of an example.

(Refer Slide Time: 11:37)



So, for this example, I consider a two layer neural network. So, the MLP is a multilayer neural network out of that I am on considering two layer with one output layer. I have

two nodes in the output layer, one hidden layer and one input layer. And we had said earlier that the input layer nodes actually gives you the identity functions or identity mapping, that means, whatever is the input to a node in the input layer that node simply outputs that input and fits it to the nodes in the next layer. The hidden layer nodes as we seen earlier actually impose the nonlinearities. So, as we increase the number of hidden layers, we can capture more and more complex form of nonlinearity. Then finally, at the output layer node, which are actually classifying layer classifying nodes, it implements actually linear classifiers.

So, if we have the cases which are not linearly separable, the hidden layer nodes actually maps those non-linearly separable inputs, non-linearly to a space where they are linearly separable. And once they are linearly separable in a latent space, then I can have a linear classifier to classify them, so that is what the hidden layer does. And we have seen that earlier.

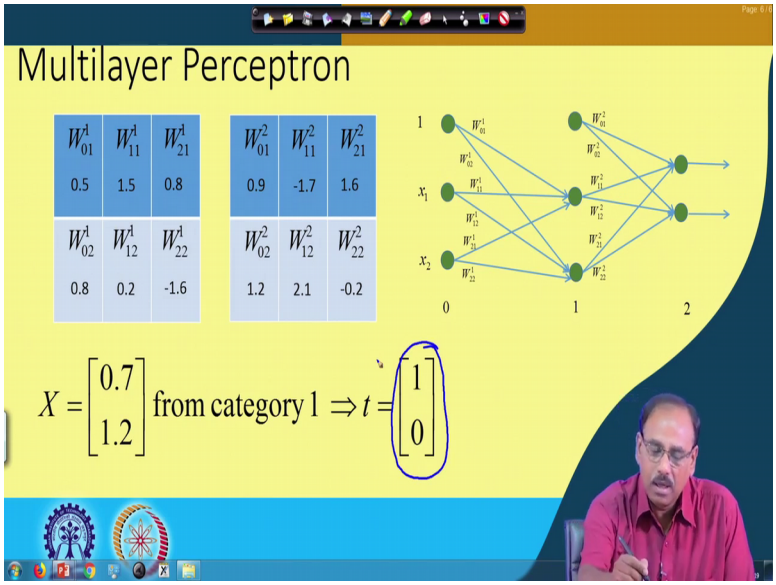
So, we compute or we consider that we have two sets of weights which later we will see that we will represent these as weight matrices. One set of weights connecting the input layer to the hidden layer, and another set of weights connecting the hidden layer to the output layer. So, those weights connecting the input layer to the hidden layer are given by W 0 1 1, because the convention that we said we are following is as I have I am considering a two layer network so the output layer is termed as layer 2; the hidden layer which is before the output layer is termed as layer 1. And the input layer is termed as layer 0.

And we have considered earlier that a node say ith node from k plus say an ith node from k minus first layer is connected to the jth node of the kth layer through a connection weight which is given by W i j k. So, following the same convention at the input layer, from the input layer to the hidden layer the connection weights or the weight values are W 0 1 1. Similarly, W 0 2 1 these are the weights which connect the zeroth node at the input layer and this zeroth node is someone is one which gives you the bias actually. So, it connects the zeroth node from the input layer to the first node and second node in the hidden layer.

Similarly, W 1 1 1 and W 1 2 1 connects the first node in the input layer to the first node and second node in the hidden layer which is layer 1. Similarly, W 2 1 1 and W 2 2 1,

they connect the second node of the hidden layer to the first and second node second node of the input layer to the first and second node of the hidden layer respectively ok. And the same the same convention is also followed for the connection weights between the hidden layer to the output layer and in this case, in this particular figure, as I am considering a two category case so my outputs are actually x 1 2 and x 2 2. So, these are the outputs which will be available from this neural network, so that the example that I am going to consider here is this.

(Refer Slide Time: 16:11)

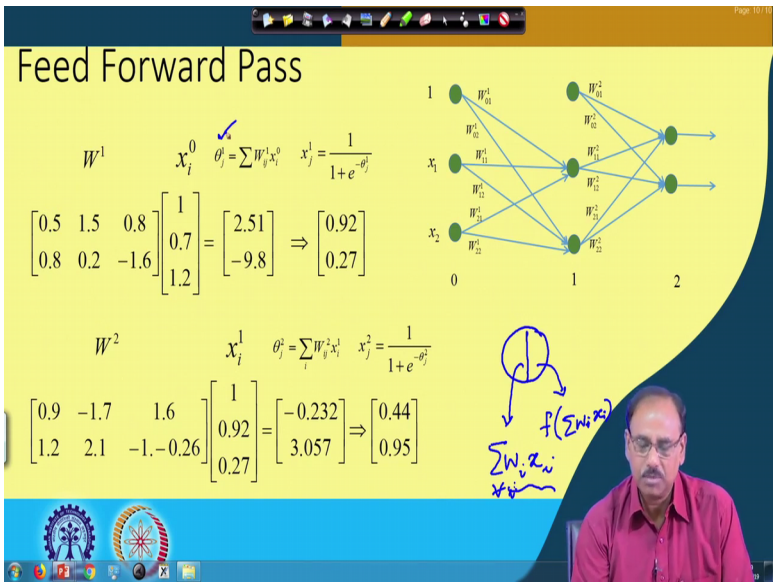


I consider a set of connection weights between the input layer to the hidden layer and hidden layer to the output layer at random initially. So, the connection weights that we given as W 0 1 1 is 0.5, W 1 1 1 is 1.5, W 2 1 1 is 0.8, then W 0 2 1 is 0.8. Again W 1 2 1 is 0.2, and W 2 2 1 is minus 1.6. So, these are the connection weights from the input layer nodes to the hidden layer nodes, so that is why we use the superscript which is 1.

Similarly, I take another set of weights W 0 1 2, which is 0.9, W 1 1 2 which is minus 1.7, W 2 1 2, which is 1.6, W 0 2 2 which is 1.2, W 1 2 2 which is 2.1, and W 2 2 2 which is minus 0.2. These are the connection weights from the hidden layer nodes to the output layer nodes ok. And I also consider an input vector which is a two-dimensional vector as given as 0.7 and 1.2. So, these are the components of the input vector. And I assume that this input vector belongs to category 1.

So, as I assume this belongs to category 1, that means, the output from the output layer nodes the output of the first node will be equal to 1, because my input is category 1. And the output of the second node in the output layer should be equal to 0, because my input belongs to category 1. So, output of the first node should be equal to 1. So, as a result the target that I have is the target vector which should be 1 0. So, the back propagation learning should try to adjust the weights in such a way that the output that I get should be equal to 1 0 or should be close to 1 0 when the learning is complete, or when the input vector is correctly classified by this neural network. So, let us see how this is done.

(Refer Slide Time: 18:59)



So, first let us see what happens in the forward pass or in the feed forward pass. In the feed forward pass, whatever you are feeding to the input as we have seen before it is processed at different layers in different hidden layers, finally, goes to the output layer and output layer gives you the final output. So, I am representing the connection weights from the input layer to the hidden layer by and weight matrix W 1 over here. So, this weight matrix W 1 represents the connection weights from the input layer to the hidden layer. So, these are the connection weights which is represented by weight matrix W 1. And simply similarly another matrix W 2 which is given, this is represented by W 2 which is representing the connection weights from the hidden layer to the output layer.

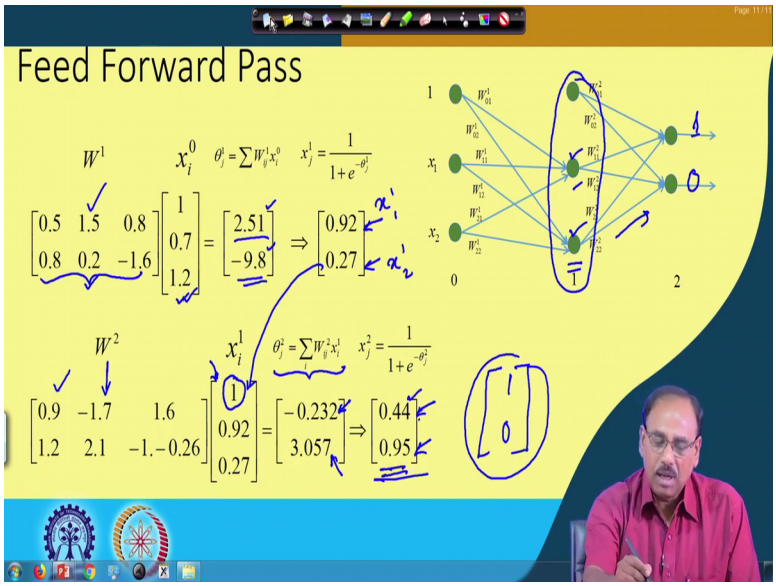
So, given these two, now we see that how this computation is done in the feed forward pass. We said that we have the input vector which is a two-dimensional vector having

values 0.7 and 1.2. So, this is 0.7 and 1.2. And as we said earlier that we append an additional component in this input vector, which is equal to 1. The whole purpose of doing this that I can have an unified representation that is the bias term which is W 0 can be considered as part of weight vector only right, so that it is W 0 plus W 1 times X 1 plus W 2 times X 2.

So, to have this unified representation, we include an additional component in the input vector X, which is equal to 1. So, given this the computation at the input layer can be done like this that the output of as we said earlier that every node in the in the neural network computes two parts. The one part computes weighted sum of the inputs which is given by W o x i summation over all i, and the second part computes a non-linear function f on sum of W i x i.

So, this non-linear function f that we are considering a sigmoidal function over here, and we are representing this weighted sum of the inputs as a different parameter or with a different variable which is theta j or which is theta j indicates from which node it comes out. So, given this, now we find that weighted sum of these two nodes in the hidden layers node 1 and node 2 will be simply given by the matrix multiplication.

(Refer Slide Time: 22:23)



I have this weight matrix W 1 multiply that or post multiply that with your input vector or augmented input vector, and that gives you the weighted sum that you get at every node in the hidden layer. So, node 1, at node 1, in the hidden layer, your weighted sum

theta 1, 1 is coming as 2.51. If you simply do this multiplication, the first row with this column vector, you get 2.51.

Similarly, the weighted sum that you get from the second node in this hidden layer is minus 9.8. If you multiply this row, second row of the weight matrix W 1 with this input vector, I get minus 9.8. So, these are these are actually the intermediate values. And final output value is after application of nonlinearity. And as I said that the nonlinearity that I am considering in this case is a sigmoidal non-linear, nonlinearity.

So, the output from the first x 1 1 will be sigmoidal function of theta 1 1. At sigmoidal function is nothing but this that is 1 upon 1 plus e to the power minus theta j 1. You remember that the superscript 1 is being used that, I am doing this computation at the hidden layer which is layer 1 in our case. So, with this the output of the first node after applying nonlinearity which is x 1 1, so this is nothing but x 1 1. So, this x 1 1 will become 0.92, and x 2 1 that is the output of the second node in the hidden layer will be 0.27.

And these are the values which are intermediate vector obtained after applying a non-linear function in the hidden layer is fade to the final output layer. So, this vector along with so this vector comes out over here appended with 1 as before to take care of the bias term as part of the weight vector. So, this is the vector which is actually fade to the input of the output layer.

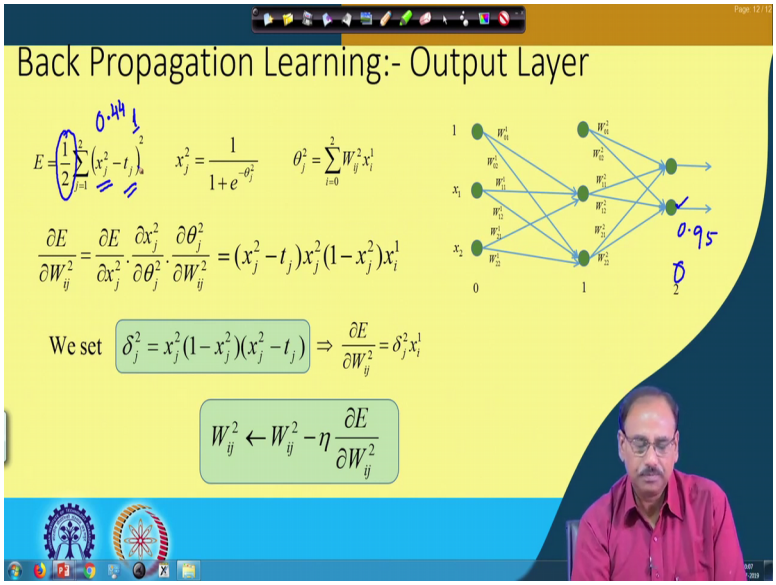
And the connection weights or the weight matrix between the hidden layer to the output layer is given by W 2 which is nothing but this. And in the same manner, the weighted sum of the inputs as given by the nodes in the output layer which is W i j 2 into x i 1, x i 1 means it is the output of the ith node in the first layer which is hidden layer. And W i j 2 is the connection weight from the ith node in the hidden layer which is layer 1 to the jth node in the output layer which is layer 2.

So, if I compute this, the product of the first row of W 2 with your input vector to output node is minus 0.232, similarly product of the second row of this weight matrix with the input vector input to the output layer nodes is 3.057. So, these are my intermediate variables. I consider this as intermediate variables as theta j 2. So, this is theta 1 2 and this is theta 2 2. Final output from this output layer is sigmoidal functions of these two quantities which comes out to be 0.44 and 0.95.

So, now find how what is the concept of error. We said that this vector X which is 0.7, 1.2, we have taken this vector x from class 1 or category 1. And because this vector x is taken from category 1, my target vector is actually 1 0, because it is taken from category 1. So, the output of the first node that should be 1, and output of the second node that should be 0 ideally or in other words output of the first node should be near to 1 very close to 1, and output of the second node should be very close to 0 if this x is correctly classified by this neural network.

But in contrast with this arbitrary or randomly chosen weight vectors, the actual output which is given by this neural network is 0.44 and 0.95. So, obviously, the output of the second node is much higher than the output of the first node. So, clearly it is a miss classification. I should actually try to get 1 0, but rather I am get in 0.44 and 0.95. So, the difference between these two vectors is what is my error. And the back propagation learning algorithm tries to minimize this error by adjusting the connection weights by propagating the error terms in the backward direction. So, now let us see that how that can be done or how the network does it.

(Refer Slide Time: 28:15)



So, these are things which we said before that my sum of squared error is given by x j 2 which is the actual output that I have got in this case this x j 2 was 0.44 if you remember or x 1 2 was 0.44 that is the output of the first neuron in the output layer; t j is the target.

So, t 1 that is the first component of the target was 1 and the first component of the output that I was getting is 0.44. So, clearly there is a difference, and that is what is the first component of the error vector. Similarly, for j equal to 2, that is output of the second neuron over here, we actually got 0.95 whereas, ideally it should be 0, so 0.95 minus 0 that is the second component of the error vector.

If squared these two error components add them together that gives you sum of squared error, I put a multiplication factor half because as I am taking sum of squared error, I have a square term when I take the derivative this square term becomes 2 and two and half gets cancelled, so that is the reason I have half over here and that is what is my loss function. The loss function that I want to minimize by using backward learning algorithm or back propagation algorithm.

Again over here you find that this x j 2, I want to minimize this with respect to different weight vectors. So, I have to go for the chain rule x j 2 is defined in terms of theta j 2 which is an intermediate variable. And theta j 2 is defined with respect to weight vectors and the output from the first layer nodes of the hidden layer notes that you get. My aim is that I should adjust the weight components W i j 2 that is the weights which are connecting the nodes in the first layer to the output layer by using back propagation algorithm.

So, for doing that what I have to do is, I have to get that derivative of this error with respect to i j 2, and then I follow the gradient descent procedure. So, I have to take the gradient of the error with respect to the weight vectors. And again here, here what we have done before, this is not new. We have discussed this before that I have to compute del A, del E, del W i j 2, and following chain rule this becomes del E del x j 2 because E is directly visible to x j. Then del x j 2, del theta j 2, because x j 2 is a function of theta j 2 and then del theta j 2 del W i j 2 because theta j 2 is a function of W i j 2.

And if you do this each of these terms del E del x j 2 is x j minus t j del x j 2 del theta j 2 is x j 2 into 1 minus x j 2, and del theta j 2 del W i j 2 is nothing but x i 1. And if you remember what we did before is we have defined this term, this part that is x j 2 into 1 minus x j 2 into x j 2 minus t j as delta j 2, because this is a term which will be passed backward to the previous layers.

And once we define this way, then my derivative del E del W i j 2 that simply becomes this term which is delta j 2 into x i 1. So, this is delta j 2 x i 1. And once I have this del E del W i j 2 my backward learning algorithm simply becomes for adjustment of W i j 2 is W i j 2 is updated as W i j 2 minus eta times del E del W i j 2 or this is nothing but del E eta times delta j 2 x i 1, where this eta is an hyper parameter which controls the rate of learning or the rate of convergence. So, in this lecture, let us stop here today. We will continue with this in our next lecture.

Thank you.