

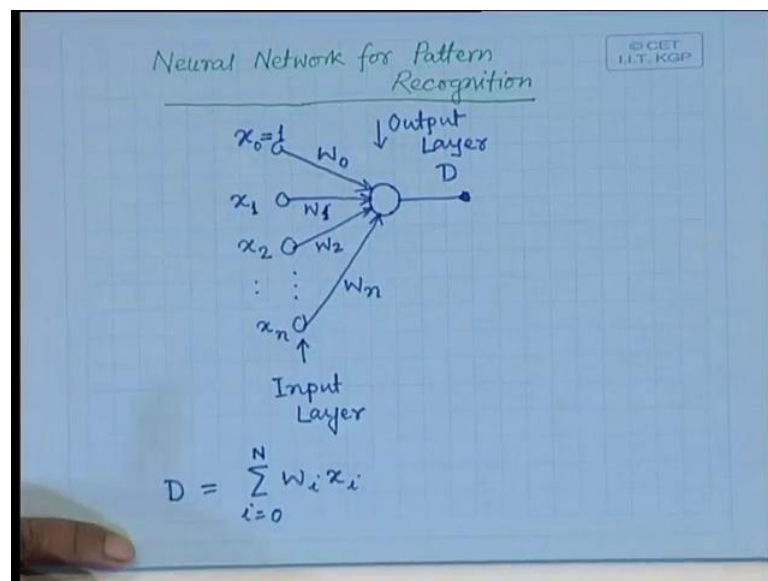
Pattern Recognition and Application
Prof. P. K. Biswas
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecturer - 25
Neural Networks for Pattern Recognition
(Contd.)

Good morning, so we are going to continue our discussion on neural network for pattern recognition. So, firstly I will just briefly go through, what we have done in our last class and then we will proceed further. So, in the last class we have talked about a single layer single output neural network, where when I say it is single layer, basically there are two layers. One of the layers is the input layer and the nodes in the input layer are simply used to connect the feature vectors.

So, the purpose of this input layer node is whatever is inputted to the input layer nodes, those nodes simply forward that input to the layer above it. And those inputs get weighted by the corresponding connection weights connecting the input layers to the layer above it, and summed up at the next layer and on top of that you apply some of sort of non-linearity.

(Refer Slide Time: 01:23)



So, effectively the single layer, single output neural network, we have said was something like this, that it has a number of input layer nodes. So, this is the layer which

is input layer. So, the number of nodes in the input layer is same as the dimensionality of the feature vectors, last one which is used to take care of the bias weight. And the number of nodes in the output layer in this case will be only one, which will indicate either one of the two classes.

So, if the output of the output layer node is greater than threshold in which is the output of the output layer node will set equal to one, which will indicate say plus one and if the output is zero, it will indicate plus 2. So, the connections that will have is something like this. So, here I have x_0 , which is usually set to 1 and the connection weight over here is the W_0 , which takes care of the bias word. Whereas, the other inputs will have a different feature components x_1, x_2 up to say x_n and the corresponding connection weights are W_1, W_2 up to W_n .

And output of the output layer node, which later say that this output is equal to D . So, this is my output layer node, where D will be equal to the weighted sum of the input feature vector components. So, it will be simply summation of $W_i \times x_i$, where i varies from 0 to N , where N is the dimensionality of the feature vector and x_0 which is usually set to 1, that is to take care of the bias weight.

So, then what we said is while training the neural network, you have to decide this connection weights or what will be the weight vector that will classify the unknown future vectors between one of the 2 classes, say class ω_1 and class ω_2 . Now, while training because all the future vectors which are given, we know what is the class belongingness of those feature vectors. So, we effectively know that what expected output is.

So, if I take training sample, a training feature vector which we know that the training feature belongs to class ω_1 , we know that the corresponding output as to be equal 1. If I know that the future vectors belong to class ω_2 , then it is known that the corresponding output has to be equal to 0. So, for that any sample or for each of the sample which is presented to the neural network, we know what the corresponding output is.

And because initially, the weights are not set properly because we start with arbitrarily set weights of very low value. So, even if our training sample which belongs to the class ω_1 is given to the neural network, the output may be 0 or output may have any of

the continuous value between 0 and 1. So, it may not be equal to 1, so what is the expected output and what is the actual output that you get the difference between that between these 2 is the error. So, if I say that my actual output that I obtain after inputting attaining example is given by the expected output is d and the actual output that I get capital, which is nothing but the summation $\sum_{i=0}^n W_i \cdot x_i$, i value is from 0 to n .

(Refer Slide Time: 01:23)

© GET I.I.T. KGP

$$D = \sum_{i=0}^n W_i \cdot x_i$$

$$E = \frac{1}{2} (D - d)^2$$

$$\frac{\partial E}{\partial w_i} = (D - d) \cdot \left(\frac{\partial (D - d)}{\partial w_i} \right)$$

$$= (D - d) \cdot x_i$$

$$\frac{\partial}{\partial w_i} (D - d) = \frac{\partial}{\partial w_i} D = \frac{\partial}{\partial w_i} \left[\sum_{i=0}^n W_i \cdot x_i \right]$$

$$= x_i$$

And then using the difference between these two, I can define what the error is. So, if I define the squared error, the squared error is given by E is equal to half of D , which is the actual output minus the expected output or which is also called as target output. So, capital D minus d whole square this is my squared error. So, while training the neural network, the aim is that we should be able to select the weight vectors of connection weights in such a way that, this sum of squared error will be minimised.

So, for that again we can make use of the gradient descent procedure. So, for making use of the gradient descent procedure, you have to find out what is the gradient of the error with respect the weight vectors. So, what I have to find out is $\frac{\partial E}{\partial w_i}$ for a particular weight vector. And if I compute this term, you find that this is nothing but if I take the derivative of this with respect to w_i , this will be simply d capital D minus d into the $\frac{\partial}{\partial w_i}$ capital D minus d upon $\frac{\partial}{\partial w_i}$.

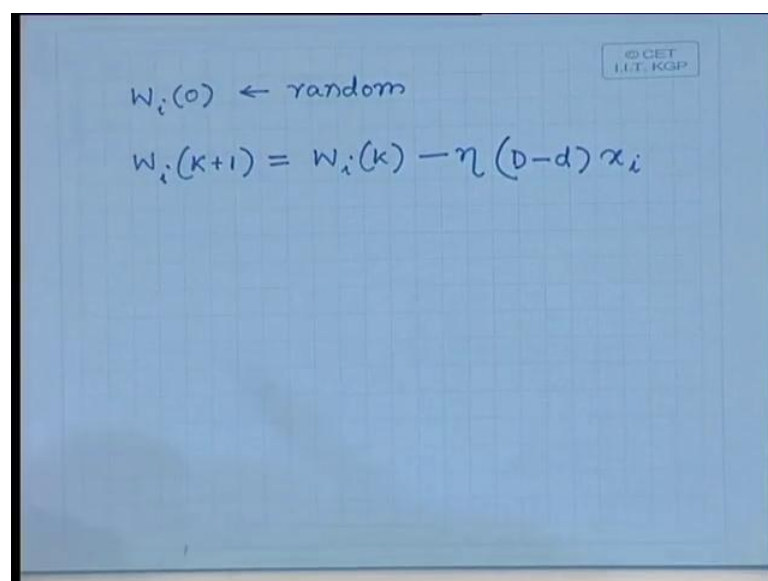
So, it is capital d minus d into $\frac{\partial}{\partial w_i}$ of capital D minus lower case d . So, if I just concentrate on this particular, because there was question on how to indicate this. So, if I

concentrate on this particular term and you find that this work is this. So, what I am trying to find out now is, what is $\frac{\partial}{\partial W_i}$ of capital D minus lower case d. So, as I said this lower case d, this is our expected output or target output so that is fixed. Given a training sample as I know what the class belongingness of training sample is so my target output is fixed, this does not depend upon the W_i .

So, this derivative $\frac{\partial}{\partial W_i}$ into capital D minus lower case d, this is nothing but $\frac{\partial}{\partial W_i}$ of capital D because lower case d is independent on W_i . And this capital D is nothing but summation of $W_i x_i$, where i varies from 0 to capital N. So, find that if I take the derivative of the summation with respect to W_i , then only term that remains is nothing but x_i because all of the terms are independent of W_i , only W_i times x_i that is dependent only on W_i and if I take the derivative it simply becomes W_i .

So, that is why this $\frac{\partial E}{\partial W_i}$ that we have obtained as capital D minus lower case d into x_i . So, only this simple derivative expression this has given me this particular expression. So, while the training the neural network as our aim to set the weight vectors. So, we have to go for a weight adjustment of that is what is neural network training? So, in this weight adjustment problem initially will have to start with arbitrary weight vectors. So, you select with some random weight vectors for different W_i , you select some random values and the values are quite small. So, you generate small random values which are taken as W_i or I generate a random web vector.

(Refer Slide Time: 10:58)


$$W_i(0) \leftarrow \text{random}$$
$$W_i(k+1) = W_i(k) - \eta (D-d) x_i$$

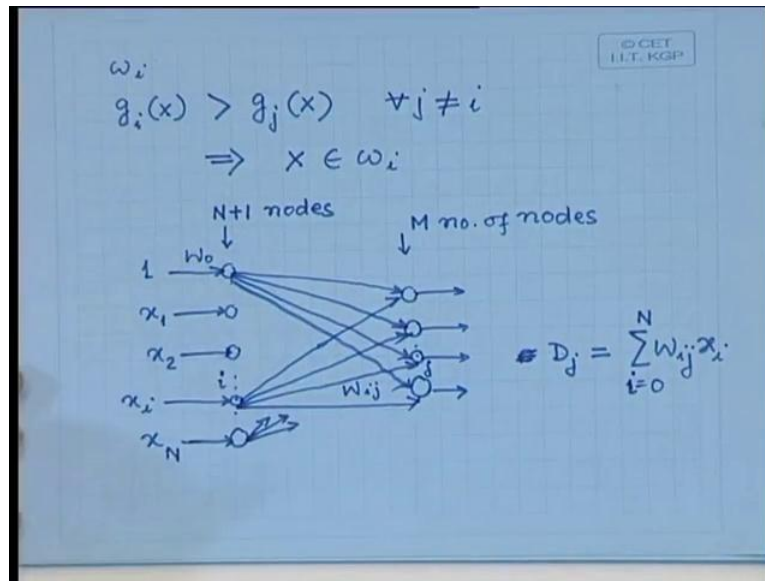
So, I have that W_i^0 , that is the initial weight vector which is set randomly, but very small value. And then in every iteration W_i^{k+1} will be some update on W_i^k that was the value of W_i in the previous iteration. So, this will be $W_i^k - \eta \frac{\partial E}{\partial W_i}$, which is nothing but $d - x_i$. So, this will simply be $d - x_i$, this is my update rule. So, how to train the neural network, you feed the training vectors one after another.

For every training vector, you find out what is this component $d - x_i$, which is the input vector component and this W_i^k was the previous vector. So, modify the previous weight according to this update formula, so that you do in the process every sample, with every sample for the weight will be modified. So, as the weight is modified for every sample, I do not know whether if you sample for which the weight was modified, because after modification with the previous unclassified sample, vectors are going to get modified.

So, it may happen with the previous sample either the previous sample will be properly classified or it will be wrongly classified. So, if it is wrongly classified I have to modify the weight vectors again. And this iteration is continued, until and unless either all the training vectors are properly classified or I come to a situation where the error that is $d - x_i$, that error is quite small. So, the convergence criteria or the termination criteria of this iterative algorithm will be that either, in a complete pass all the training samples are properly classified or I get an error or the error is big or small, the error which I can tolerate.

So, that will be my termination criteria or convergence criteria for this training algorithm. So, with the single layer single output neural network, we have considered only two class problem, that is if the output is greater than threshold or if the output is greater than zero, I say that sample belongs to one class. If the output is less than zero, I say that the sample belongs to another class. So, the kind of thresholding which we are applying is a hard threshold, but when you have multiclass problems, naturally the multiclass problems cannot be taken care of by a single output. So, I have to have multiple outputs. So, that gives you multiple output single neural networks and the action of this will be similar to all linear machine.

(Refer Slide Time: 14:35)



In case of linear machine, we have said that for every class ω_i , I have a discriminant function g_i of x and we say that if g_i of x is greater than g_j of x for all j not equal to i . So, this indicates that the feature vector x belongs to class ω_i , that is for every class, I have a discriminant function for the given unknown feature vector x , you compute the discriminant function for every class. And whichever class gives you the maximum value of the discriminant function, you assign this unknown feature vector x to the corresponding class.

So, if I want to have this similar kind of action using this neural network, then obviously I have to have a neural network with multiple numbers of outputs. So, the input layer as before will remain same that is the number of neurons in the input layer will be same as the dimensionality of the feature vectors of class 1, so that additional node takes care of the bias weight. And in the output layer, we have a number of nodes, which is equal to the number of classes that we have say if there are m numbers of classes, then in the output layer I have to have m number of nodes.

And here I have n plus one number of nodes and n is the dimensional to open. So, you apply the feature vectors to this inputted nodes, as you have assume the first component is equal to 1, that takes care of the bias weight w_0 other components are x_1 x_2 up to here have its x_i and here it is x_n . Now, the connection is like this that for every inputted node, the output of a every node will be connected to the input of every node in the

output. So, the connection will be like this. Let us consider one node so here this is j node.

So, similarly it will be like this. Similarly, this will also be connected to every node in the output here. And these are the outputs from every output of the output layer node. Now, the output of the j node suppose, I call it j th node in the output layer. Output of the j th node in the output layer will be nothing but if I represent this by saying D_j , which will be equal to some of $W_{ij} \times x_i$, where i varies from 0 to capital M . And what is W_{ij} ? And W_{ij} is the connection weight from the output of the i th layer in the input node to the input of the j layer in the output node.

So, when this is my i th node in the input layer and this is the j node in the output layer, this connection weight from the i th node in the input layer to the j th node in the output layer is W_{ij} . So, given this sort of connection and the sort of network topology, here training of the neural network means that I have to select all the connection weights W_{ij} for all values of i and all values of j . And unlikely in case of single layered network and single output neuron at the output was of scalar value, it was not a vector, it was a single value either 0 or 1.

In case of multiple outputs, the output will not be a scalar, but the output will be vector because I have capital M number of components. All the output layer nodes will give me some output, so the output was also a vector not a scalar. However for unknown feature vector that is during training as well using the entire feature vector belonging, this is known. So if I know that a feature vectors belongs to class ω_3 , I know my target output should be only the hard neuron in the output layer should give me a value 1. All other neutrons and output layer should give me value 0.

If the feature vector belongs to class ω_1 , then only the first output layer neuron should give me an output 1, all other output layer neurons should give value 0. So, my target feature vectors will have only one of the components to be equal to one, the rest of the components should be equal to 0. However, as initially the weight vectors are not properly set, I may get an output vector, which is not the same as target vector. So, accordingly the error that I get is affecting the difference between 2 vectors, the target vector and the actual vector.

(Refer Slide Time: 21:25)

The image shows a handwritten derivation on a blue grid background. At the top right, there is a small logo for '© GET I.I.T. KGP'. The main content consists of the following equations and text:

$$E = \frac{1}{2} \sum_{j=1}^M (D_j - d_j)^2$$
$$\frac{\partial E}{\partial W_{ij}} = (D_j - d_j) \cdot x_i$$

$$W_{ij}(0) \leftarrow \text{random}$$
$$W_{ij}(k+1) = W_{ij}(k) - \eta (D_j - d_j) \cdot x_i$$

Training Algorithm.

So, accordingly we have to define the sum of squared error, which in this case, we can define as E is equal to half of D_j minus lower case d_j . So, this D_j means the actual output of the j th neuron in the output layer or else this lower case d_j , this indicates that what should be the value of the output of the j th neuron in the output layer in my target vector. And because I have M number of components in output vectors, so the sum of squared error will be simply this square, take the summation for j is equal to 1 to M as I have M number of components in the output vector.

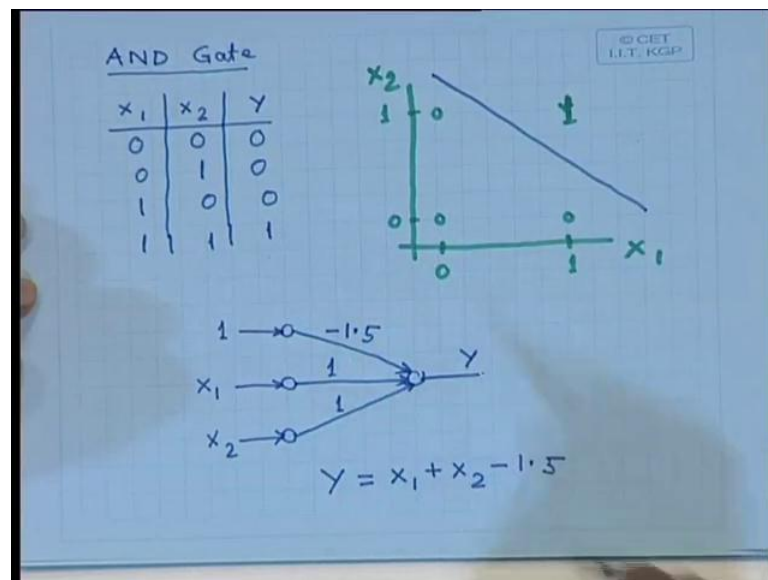
Now, given this, here again you find that for the training algorithm or for learning of the neural network I have to take the gradient of this with respect to W_{ij} . Again following the similar derivative to procedure I can find out that this $\frac{\partial E}{\partial W_{ij}}$ will be simply being D_j minus d_j into x_i . Our x_i is the i th component of the input feature vector. I will get the same, with following the same procedure I get this. So, once I get this, now my training algorithm before will be $W_{ij}(0)$, that is the initial weight.

You have to set arbitrary or this will be random, but a small value and while iteration, this $W_{ij}(k+1)$ will be equal to $W_{ij}(k) - \eta (D_j - d_j) \cdot x_i$, which is nothing but that of convergence. This result set of convergence into D_j minus d_j , which is an error time x_i . So, this is what the Training algorithm is. So, as before the training algorithm being an iterative procedure, I have to decide what are the convergence criteria. So, again the convergence criteria can be altered, all the training samples are correctly classified by a

particular set up with vectors or I reached a condition at the electrons small that is the error is tolerable.

Now, if you look at the disc made function that this kind of neural network actually imitates, this is the linear disc made function, because it is nothing but sum of $W_i x_j$ for all values of i varies from 0 to M . So, it is a linear equation for fixed weighed vectors. So, that means for every neuron in the output layer node, actually gives me linear equation of the straight line or an equation of a hyper plane. So, that simply says that if the classes are linearly separable, I can design the decision boundary using this sort of neural network, but if they are not linearly separable a single layer neural network cannot give me the decision boundary.

(Refer Slide Time: 25:53)



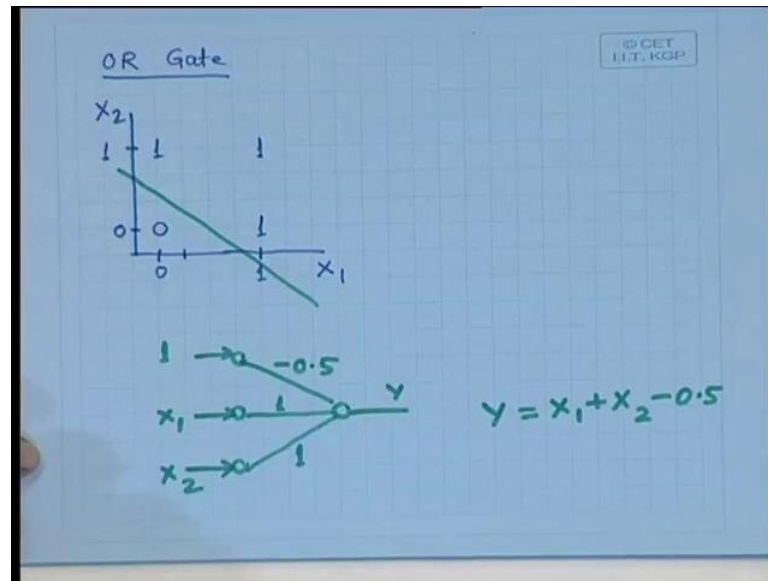
So, let us take some few examples from say binary arithmetic. So, let us take the case of an AND gate. So, all of you know the truth table of an AND gate. If I have two variables x_1 x_2 and I have this output y , the truth table of the AND gate, it is simply 0 0 0 0 1 0 1 0 0 and 1 1 1. I can consider these to be a 2-dimensional binary feature vector, x_1 x_2 to be 2 dimensional binary features, which can assume one of these four different values. And I want to design a neural network, which will give me the function of an AND gate, that means output should be equal to 1, only when both the feature components are equal to 1 and output should be 0, if any or both of the input feature components are equal to 0.

So, if I plot it in the form of a graph something like this. This is my x_1 dimension, this is my x_2 dimension and this is the origin. So, for $(0, 0)$ I take 0 some over here, this is 0 and somewhat here it is 1. So, if x_1 is 0 and x_2 is 0, so the output should be equal to 0, if x_1 is 0 x_2 is 1, then also output should be equal to 0, if x_1 is 1 x_2 is 0 then also output should be equal to 0, if x_1 and x_2 both of them are 1 then output should be equal to 1. So, find that given this sort of the outputs, I can define a boundary line somewhere over here.

So, this is my classifier and given the sort of classifier, you find that I can have a single layer, single output neuron network, which will give me this sort of decision boundary. And one of the possible neural networks that can give me the sort boundaries is like this. I will have three inputs and one output, so one of these three inputs will have input is equal to 1 and the other inputs will have my feature components x_1 and x_2 . The connection weights can be something like this, I put minus 1.5 here, 1 here and 1 here and this is my output y .

So, find that this y actually, gives me the equation of the straight line which is nothing but $x_1 + x_2 - 1.5$. So, here find that if any of the components or both of the components are 0, output is minus 1.5. If any of the components is 0 the other one is 1, then the output is minus 0.5. Now, in all three cases output is less than 0, if both this component are 1 then output is 0.5 which is greater than 0. So, whenever this y is less than 0, I can set output to be equal to 0, whenever y greater than 0, I can set output is equal to one. So, this simple neural network can perform the function of AND gate.

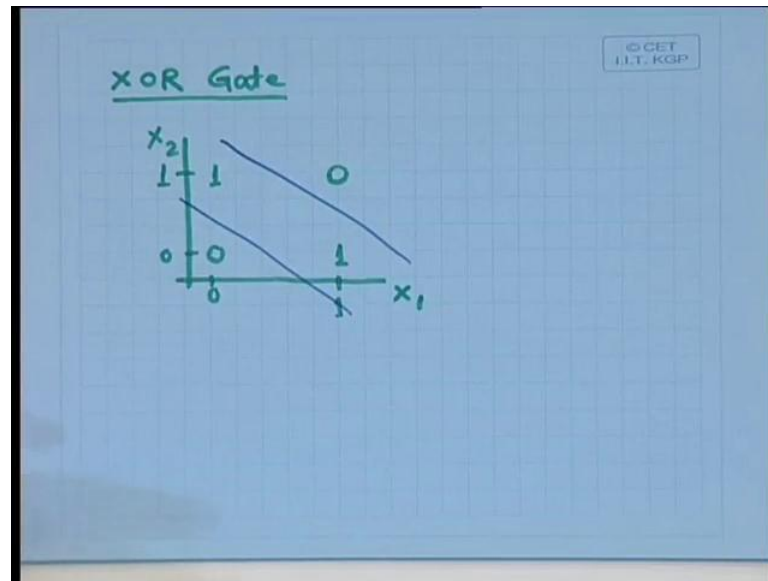
(Refer Slide Time: 30:32)



Similarly, if I take another example that is an OR gate. In case of an OR gate, the similar plots should be something like this, that is this is x_1 , this is x_2 . I put 0 here, 0 here, 1 somewhere here, 1 somewhere here. So, in case of OR gate, you find that only when both the inputs are 0, output is 0; if any of the input is 1 or both the inputs are 1, in that case output is 1. So, I have 1 here and 1 here and I have one here. Again in this case, I can define a straight line, which will separate between these 2 regions. And again a simple neural network like this with three input layer nodes and one output layer node, but this y again gives me linear equation, which are nothing but x_1 plus x_2 minus 0.5.

So, here you find that both x_1 and x_2 both of them are 0, the output is minus 5 which is less than 0. In all other cases if any or both of them becomes 1, the output becomes greater than 0. So, following the same logic if the output is greater than 0, I put it to 1, if the output is greater than 0, I put it to 0. Then to find that this particular neural network will give me an OR function.

(Refer Slide Time: 32:50)



Now, if I take another gate of similar nature that is X-OR gate. In case of X-OR gate, if I put similar functions x_1 x_2 , I put 0 here, 0 here, 1 here and say 1 here. In case of X-OR gate if the inputs are 0 0, the output is 0, if the input are 1 1 then also output is 0, if the inputs are 0 1 or 1 0 then only output is 1. So, I have a 1 here and I have 1 here. Now, given this sort of situation, you find that I cannot draw a single straight line which will separate this space where in one of the half space, the output 0 and the other half space, the output is 1.

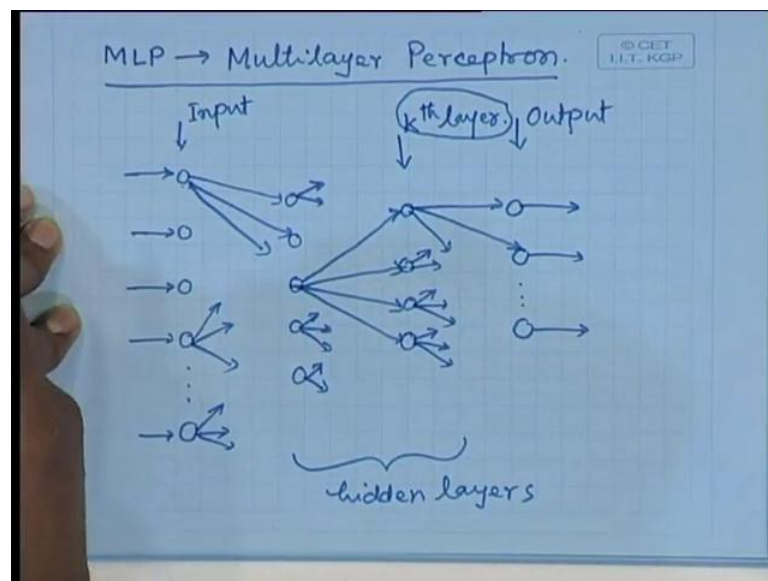
Effectively, what I need is I have to define two straight lines here so that I can put the 0 output in one region and the output 1 to be in another region. So, this is very simple example and I cannot have the equation of a single straight line, which can divide this space into such a sort of region. So, naturally of single layer neural network cannot give me a solution for a simple problem something like this, of its other region cannot be divided into 2 half spaces using linear equation.

So, exactly for this kind of situation or the region is not linearly separable, I have to go for multilayer neural network, the single neural network does not work for this set of spaces. So, I had to have multiple, a neural network are one of the layer, obviously the input layer, which we do not consider in deciding the number of layers and the other layer is the output layer. In case of single layer, we have the output layer and in between I have to have a number of hidden layers.

And it is observed that if you have 2 hidden layers, 1 or 2 hidden layers, two at the most that is sufficient. Most of the practical problems should go for multilayer neural network with one output layer and 2 hidden layers or if the problem is simpler, we can have one output layer and one hidden layer. The number of nodes in the input layer will be same as the dimensionality of the feature vector plus 1, the number of nodes in the output layer will be same as the number of classes, for which we are designing the classify.

And the number of nodes in the hidden layers those are actually variable. And so far there is no formal way in which you can decide that how many number of nodes in the hidden layer, which we have to have that type of problem. So, it depends upon how complex the decision boundaries are among the different classes. So, in case of this multilayer neural network, this is also called multilayer perceptron or MLP.

(Refer Slide Time: 36:32)

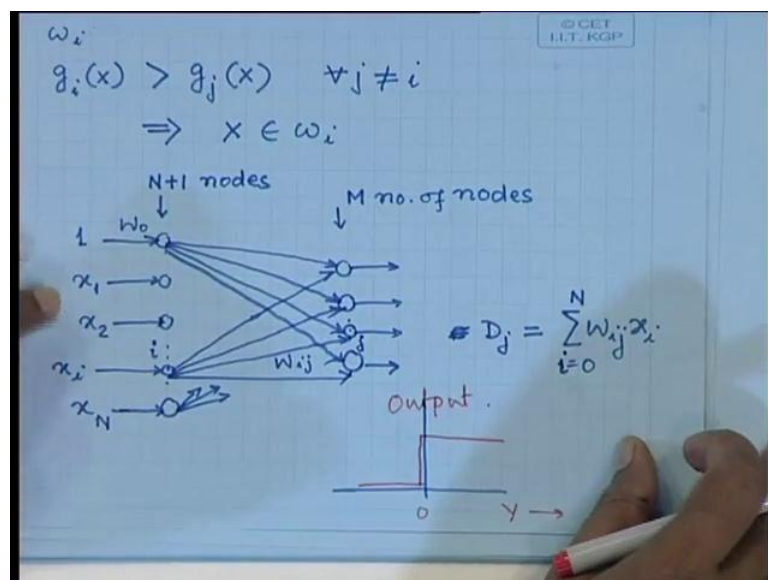


This is multilayer perceptron, we have obviously one layer which is input layer, we have one layer which is the output layer and we have either 1 or 2 hidden layers like this. So, this is my input layer, this is the output layer and these 2 are the hidden layers. So, as before the outputs from the k th layer goes to place of the $k+1$ th layer. And every neuron output of every neuron in the k th layer is connected to the input of every $k+1$ th layer, so I have putted like this.

So, similarly here all of them are connected. Similarly, here and finally, you get the output from the output layer nodes. So, when I consider any layers the k th layer. Now,

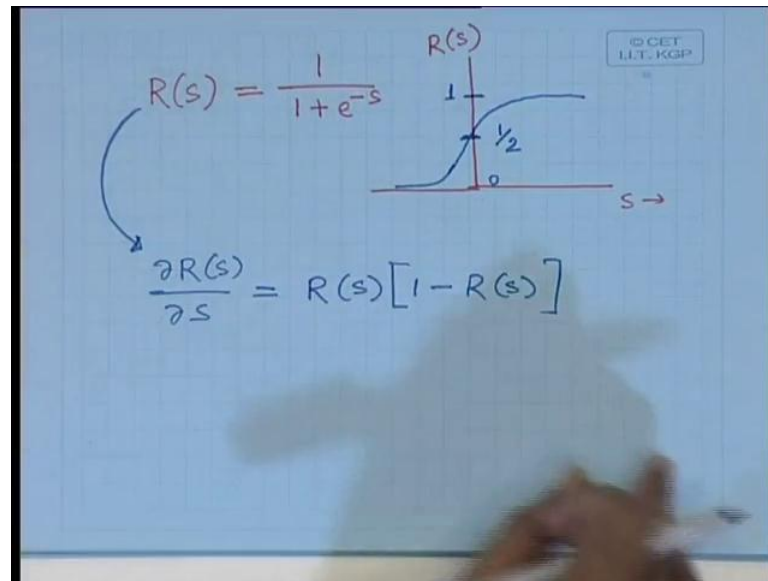
find that the number of weight vectors that we have is much more than the number of weight vectors that we had in earlier classes, because from the nodes of every layer to the nodes of the upper layer is set of weight vectors. And the number of weight vectors of the sets of weight vectors that we have depends upon how many layers you have in this neural network. So, if I consider say node, and not only that each of the nodes, in all the layers except the input layer gives a non-linear function, employs a nonlinearity, which earlier this week said that, the small layers nonlinearity was hard non-linearity or threshold linearity. That is, over there are non-linear function in this case was something like this.

(Refer Slide Time: 39:58)



The non-linear function that we had also of this form that is, if y is less than 0 then output is 0, if y is greater than 0 then output is 1. So, this was also a non-linear function, but this nonlinearity was harder nonlinearity. Whereas in case of multilayer neural network, the nonlinearity which is used is not a hard non-linearity, but it is sort of sigmoidal function.

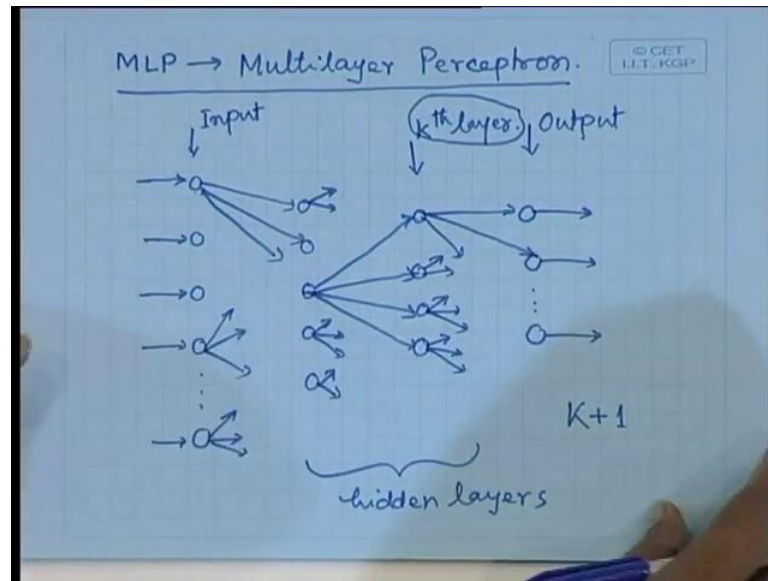
(Refer Slide Time: 40:46)



The sigmoid function is given by this expression, say if s is a argument, then r is which is a sigmoid function is 1 upon 1 plus e to the power minus s . And if you plot this R s verses s , this is the s and this is R s , then the plot will b like this. You find that when s tends to minus infinity, R s tends to be 0 because as s tends minus infinity, this e to the power minus s that tends to be infinity. So, 1 upon this term tends to be 0. Similarly, as s tends to class infinity, then this R s tends to be equal to 1.

So, I have the output function which is limited between 0 and 1, as s tends minus infinity R s tends to be 0, as s tends to minus infinity R s tends to be 1 and at s is equal to 0, the value of R s is equal to half. So, it is a symmetrical non-linear function around the values s is equal to 0. The advantages that we get by using this type of nonlinearity are that, this function is differential. In case of hard nonlinearity because I have a discontinuity act is equal to 0, it is not differentiable. As this function is a continuous function, it is differentiable and not only that, if I take the differentiation of R s del R s del s , that gives us a very simple form, this is nothing but R s into 1 minus R s . If you differentiate this R s with respect to layers will sort of differential output.

(Refer Slide Time: 43:09)



So, here I assume that in this multilayer neural network, I have total capital K plus 1 number of layers. One of them is input layer and I have capital K number of layers including the hidden layers and the output layers. So, the total number of layers is K plus 1. And when you say, we say that it is K layer MLP, where k indicates the number of layers excluding the input layer.

Now, given this sort of situation, I have to when I talk about the input to a particular node or when I talk about output to a particular node, I also have to identify that which layer it is. So, I have to identify which layer and at the same time, which neuron in that particular layer. So, if I say the output of the K th layer neuron or say the i th node in the k th layer neuron.

(Refer Slide Time: 44:22)

© CET
I.I.T. KGP

$$x_i^{(k)} \rightarrow \text{output of } i^{\text{th}} \text{ neuron in } k^{\text{th}} \text{ layer.}$$
$$x_j^{(k+1)} = R \left(\sum_{i=0}^{M_k} W_{ij}^{(k+1)} x_i^{(k)} \right)$$

Then that will be represented by $x_i^{(k)}$, this is output of i th neuron in the k th layer. Let me put it as lower case k , so it is a variable. So, we have said that capital K is number of layers. So, $x_i^{(k)}$ as K superscript and i subscript, this represents output of i th neuron in the k th layer. And obviously this output is straight as an input to a node in the k plus plus layer. So, when I talk about the output of the k plus plus layer and the output of the j th neuron in the k plus plus layer, that I can write as $x_j^{(k+1)}$.

This is output of the j th neuron in the k plus plus layer, which gets input this sort of input for different values of i . So, this I can write as $W_{ij}^{(k+1)}$ because every i th node in the k th layer, this j node in the k plus plus layer gets input. So, the $W_{ij}^{(k+1)}$ and between which 2 layers so I write it as $k+1$ indicating that it is the connection weight between the k layer neuron and the k plus plus layer neuron. So, $W_{ij}^{(k+1)} x_i^{(k)}$ that is output of i th node in the k layer neuron.

Take the summation of this, for i varying from 0 to let us put it is as M_k indicating, I have M_k number of nodes in the k th layer. And i is equal to 0 indicates that, if I want to incorporate any bias in the k th layer also, I can have bias only in the input layer, I can also have bias in every layer as well. So, M_k is the number of nodes in the input layer and plus 1, so the total number of nodes in the input layer is $M_k + 1$. So, i set 0 is equal to M_k and on top of this, I have to have this non-linearity.

So, the whole thing will be a non-linear function of this, so R of this. So, output of the j th neuron in the k plus plus layer is the weighted sum or non-linear function of the weighted sum of outputs of all the neurons in the k th layer where, this weight is given by the corresponding connection bias j k plus. And this is valid for all the hidden nodes and output nodes. So, every node in every layer gets outputs from the nodes in the previous layer, except the input layer node.

To the input layer node with great effect, the inputs and the output of the output layer nodes, that we get directly as outputs that is not fit to the neurons of any other layers. So, given this sort of formulation, of this sort of architecture of the multilayer neural network, here again I have to find out what will be my training procedure. So, to do the training procedure, I follow the similar type of approach that is the gradient descent approach. So, for that I have to find out what is the error I get at the output of every layer.

(Refer Slide Time: 49:12)

The image shows a whiteboard with handwritten mathematical derivations. At the top right, there is a small logo for '© CET I.I.T. KGP'. The main derivation starts with the error function for layer K:

$$E = \frac{1}{2} \sum_{j=0}^{M_K} (x_j^{(k)} - d_j^{(k)})^2$$

Below this, the partial derivative of the error with respect to the weight $w_{ij}^{(k)}$ is shown:

$$\frac{\partial E}{\partial w_{ij}^{(k)}} = (x_j^{(k)} - d_j^{(k)}) \cdot \frac{\partial x_j^{(k)}}{\partial w_{ij}^{(k)}}$$

Then, the partial derivative of the neuron's output $x_j^{(k)}$ with respect to its weight $w_{ij}^{(k)}$ is derived using the chain rule:

$$\frac{\partial x_j^{(k)}}{\partial w_{ij}^{(k)}} = \frac{\partial}{\partial w_{ij}^{(k)}} R \left(\sum_{i=0}^{M_{K-1}} w_{ij}^{(k)} x_i^{(k-1)} \right)$$

$$= x_j^{(k)} \cdot (1 - x_j^{(k)}) \cdot x_i^{(k-1)}$$

So, finally, when I am saying that capital K indicates my output layer. Let us see that what is the either at the output. Every neuron except the input layer, in input layer it is unity. Just linear in an input, whatever is the input, to pass it to output, but in all other layers this nonlinearity. In case of single neutron, we said that this nonlinearity is present only in the output layers node not in the input layers node and there we have considered the nonlinearity to be hard threshold.

Here we are considering this nonlinearity to be single sigmoidal function, it is in the hard threshold. No, that is what I am saying, we will come to realise that, why this nonlinearity are putting. We can do it definitely because that is also linear, but the problem was there is if you do not put any such limit, then your network may have becomes saturated. So, that is why you put the limit at every neuron. And we are not putting the hard threshold, but we are putting this continuous non-linearity, that is will be obviously talked about the training algorithms.

Student: Sir, number of nodes in hidden layer that may vary layers to layers?

May vary from layer to layer.

Student: On what factors it may vary?

That is what I said, there is no analytical approach to find out what is how many number of layers you need or the how many nodes in every hidden layers you need, there is no such analytical factors.

Student: If we increase the numbers means the quality of output

Up to some level the quality of output will increases beyond that the quality may fall.

Student: This non linear function is different for different layers.

It can be different for different layers or it can be same. The usual practices you use the same non-linear function. What is needed is in each of this layers the nonlinearity that you introduced, the nonlinearity should be a continuous function or it should be a derivable function or it should be able to find its derivatives, its derivatives should exists, that is the constant. Why as I said that will be clear, when I talk about the training algorithms. Why we want that this function should be differentiable function?

So, what is the error, let us assume that in k th layer, as before the sum of squared error E will define as half of say x_j^k , that is the output that you get from the j th neuron in the k th layer minus the d_j . Let us put k here also, that is the target output, square of this. Take the summation for j varying from 0 to M_k and this is the number of nodes in the k th layer. Now, what is this x_j^k , this x_j^k is of this form. Here in place of k plus one, you

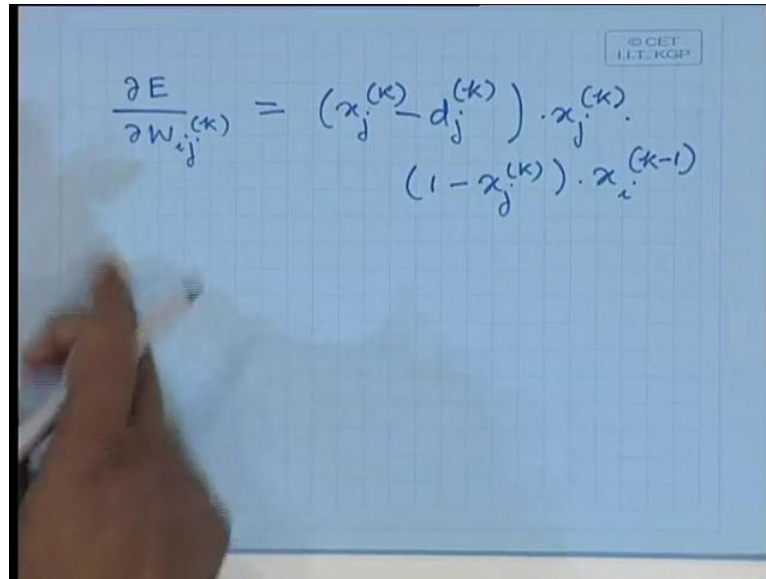
would just take k and accordingly modify this expression. So, that is what is x_{ij} , that is what I get from the j th neuron in the k th layer.

So, to apply my training algorithm problem the between this and procedure are to take the triplet individual layers with respect to do to W_{ijk} . So, what to find out is, I have to take $\frac{\partial E}{\partial W_{ijk}}$, what will be this? Again following the similar derivative approach, this will be simply x_{jk} minus d_{jk} into $\frac{\partial}{\partial} x_{jk}$ upon $\frac{\partial}{\partial} w_{ijk}$. Now, here we find that, why I am putting this $\frac{\partial}{\partial} x_{jk}$ $\frac{\partial}{\partial} W_{ijk}$, which I did not put because of single layer.

The reason is this x_{jk} is again a weighted sum of the output from the neurons from k minus first layer. And outputs of the k minus first layer are actually weighted by the corresponding W_{ijk} . So, I have to do this and not only that this x_{jk} is now on non-linear function of the weighted sum of the outputs of the k plus first layer neurons. So, when I take this gradient, this gradient has to take care of the nonlinearity as well. So, when I compute this, as I said earlier that the nonlinearity that I have is of this form R^s is equal to $\frac{1}{1 + e^{-x}}$.

And if I take the derivative of this with respect to the W_s , the derivative is like this. So, if I concentrate on this term that is $\frac{\partial}{\partial} x_{jk}$ upon $\frac{\partial}{\partial} W_{ijk}$ so this will be nothing but $\frac{\partial}{\partial} w_{ijk}$ of the non-linearity R of W_{ijk} , x_{ijk} minus one. This summation will be for i is equal to 0 to M_k minus 1 . So, if I take this directive you will find that, this will be nothing but x_{jk} into $1 - x_{jk}$, coming from this particular expression times x_{ijk} minus 1 . So, if I put this in this expression.

(Refer Slide Time: 57:14)


$$\frac{\partial E}{\partial W_{ij}^{(k)}} = (x_j^{(k)} - d_j^{(k)}) \cdot x_j^{(k)} \cdot (1 - x_j^{(k)}) \cdot x_i^{(k-1)}$$

What I get is $\frac{\partial E}{\partial W_{ij}^{(k)}}$, that will be simply $x_j^{(k)} - d_j^{(k)}$ into $x_j^{(k)}$ into $1 - x_j^{(k)}$ into $x_i^{(k-1)}$. So, this will be the derivative of the error with respect to $W_{ij}^{(k)}$. So, naturally when I go for the iterative training algorithm, I have to use this derivative for updation of the reflectors and this is become from layer to layer. And in case of single layer output, it was only one layer now I have multilayer perceptron so this weight updation has to be done from layer to layer. So, let us stop here today, we will continue it in the next class.

Thank you.