Intelligent Systems and Control

Prof. Laxmidhar

Department of Electrical Engineering

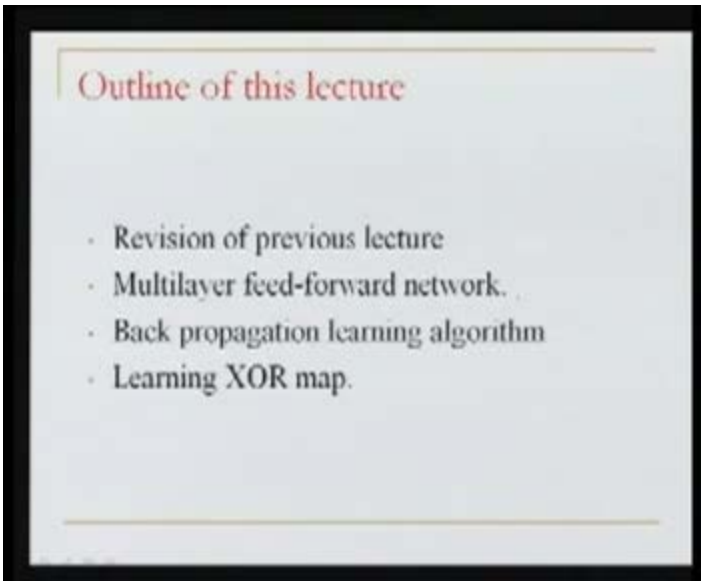
Indian Institute of Technology, Kanpur

Module-1 Lecture-2

Multilayered neural networks

This is the second lecture of module-1 on intelligent control. The outline of this lecture is like this: revision of the previous lecture, multilayer feed-forward network, back propagation learning algorithm and learning XOR map.

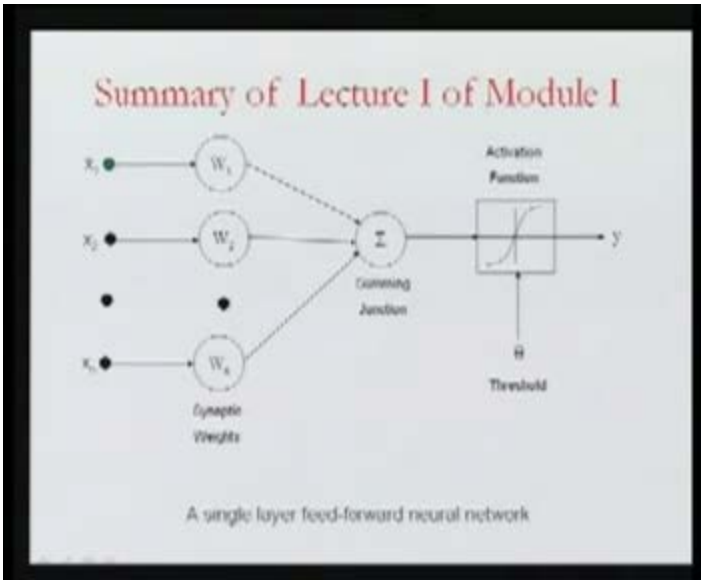
(Refer Slide Time: 00:32)



These are the topics we will be discussing today. These are the summaries that we discussed in the last class.

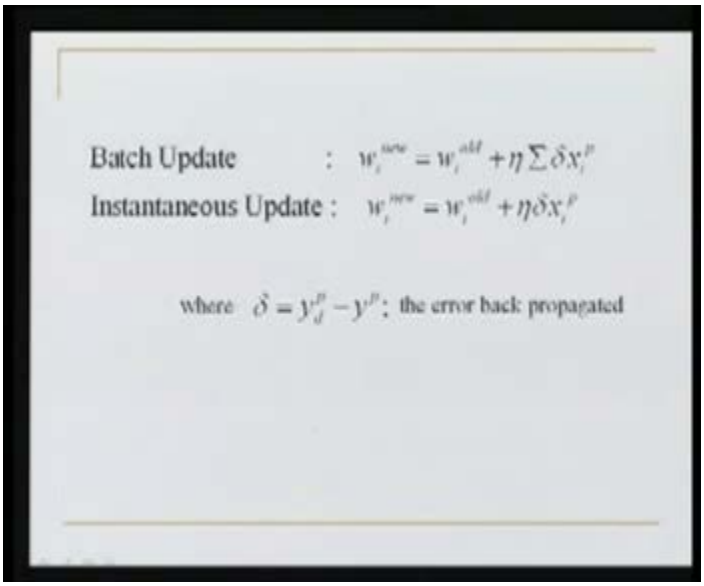
What you are seeing is a single neuron. We have multiple inputs that are multiplied with the weight associated with this connection. They are summed up in this summing junction and then you have an activation function. In linear neural network, what we consider is that this is simply a constant. This I can say is linear.

(Refer Slide Time: 00:49)



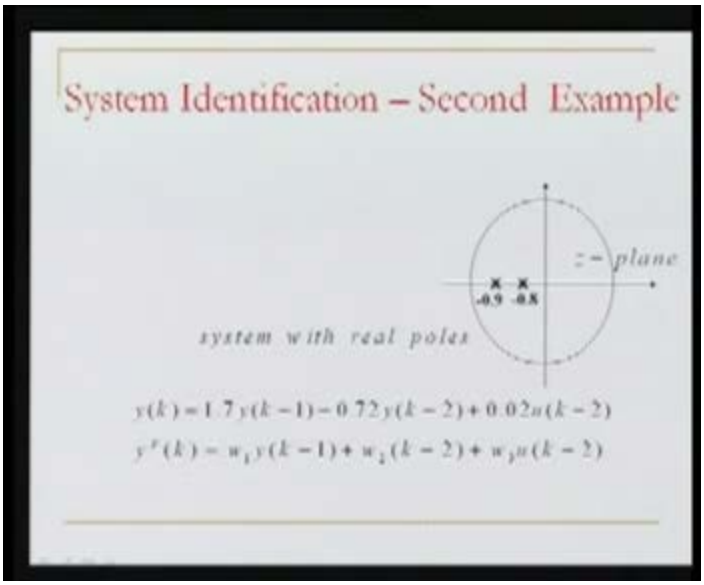
The batch update law that we said is that new weight vector is old weight vector plus eta into summation of all error terms due to all the patterns, whereas instantaneous update is done simply by the back propagation term due to a single input pattern, where delta is the error back propagated.

(Refer Slide Time: 1:15)



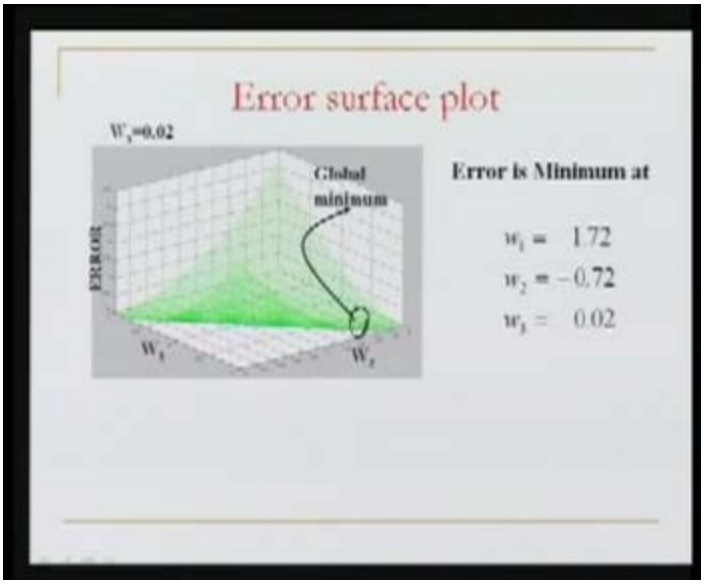
We will take another example today. This example is where we have two poles within the unity circle, but they are all real poles; minus 0.9 and 0.8. This is my system dynamics and my model is this. y p k is w_1 y k minus 1 w_2 y k minus 2 plus w_3 u k minus 2.

(Refer Slide Time: 1:41)



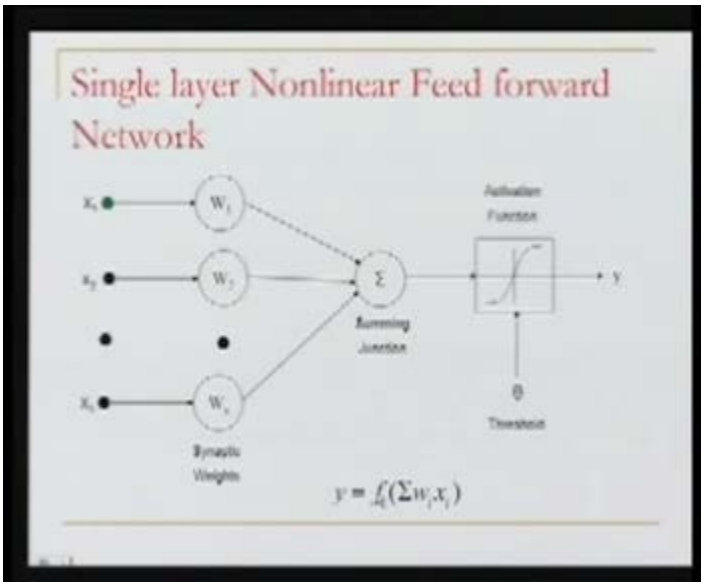
I take the same single neuron with three inputs, y k minus 1 y k minus 2 and u k minus 2. When I do that, I finally get these exact values; w_1 is 1.72. w_1 converges to 1.7, w_2 converges to minus 0.72 and w_3 converges to 0.02. This is what Error is minimum at w_1 1.72, minus 0.72 is w_2, and w_3 is 0.02. This error surface looks like this and finally, here is your global minimum.

(Refer Slide Time: 2:39)



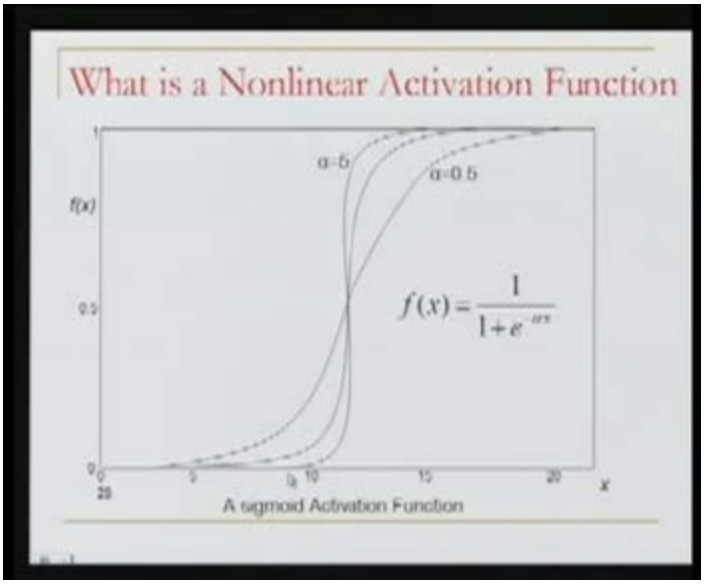
We talked about a single layer linear feed forward network. Now, let us talk about single layer nonlinear feed forward networks. In this, the activation function becomes non-linear. So, f is a non-linear function. The question arises as to what kind of nonlinear function we can consider as an activation function.

(Refer Slide Time: 3:09)



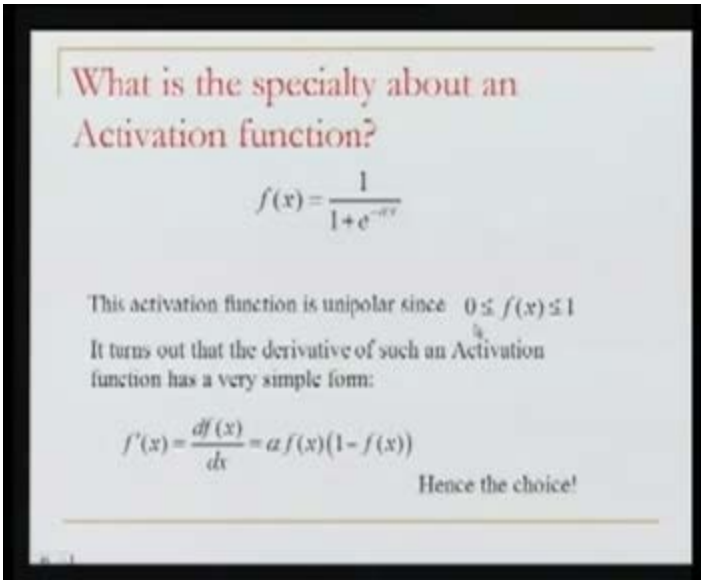
Here is a non-linear activation function. The axis is x and this is f of x and the function is f of x is equal to 1 by 1 plus e to the power of minus alpha x. By changing alpha value, we can change the transition from 0 saturation to saturation 1. The function varies from 0 to 1 and alpha determines either the sharp rise or slow rise from 0 saturation level to 1 saturation level. This is a non-linear function. Why did we select this activation function?

(Refer Slide Time: 3:22)



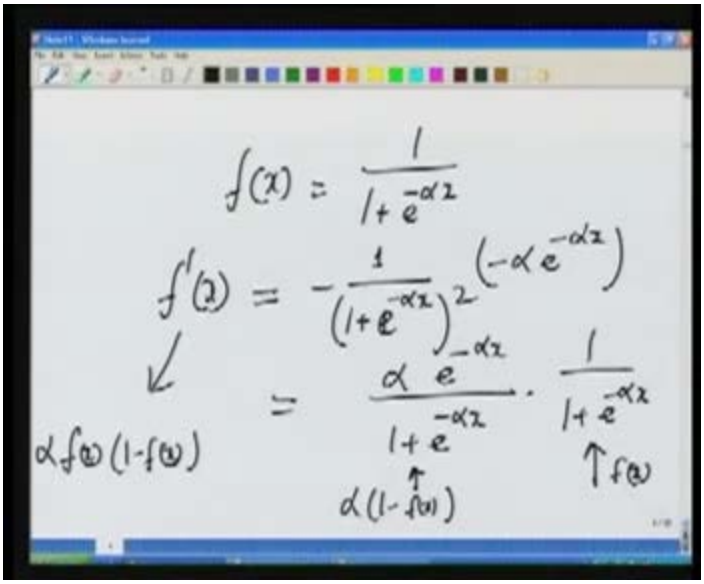
If we look at it, you may ask what is the speciality of this activation function? It is very clear from the previous graph of this function that f of x is always limited or is limited by boundary 0 and 1. f of x is less than 1 greater than 0, but the interesting part of this function is if I differentiate this function f of x, d f of x upon dx is equal to alpha f of x into 1 minus f of x. We can see that derivative of such an activation function has a very simple form.

(Refer Slide Time: 04:21)



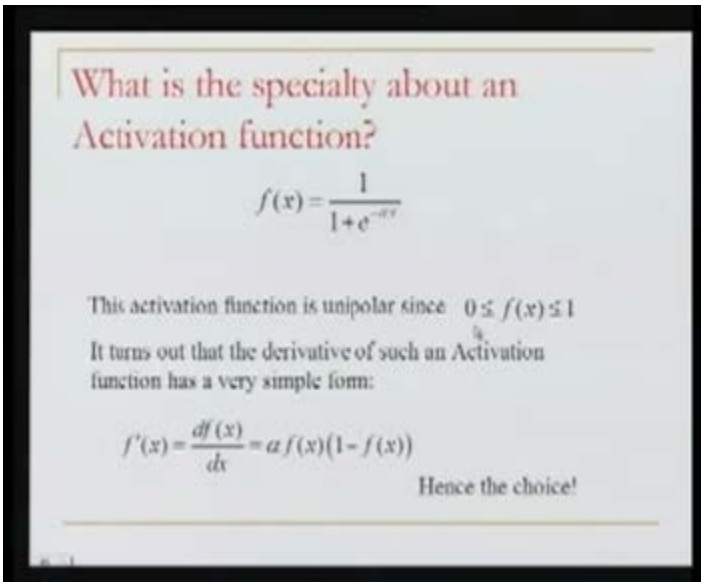
We had an activation function of this form: f of x is equal to 1 upon 1 plus e to the power of minus alpha x. I said that the differentiation of this activation function is very simple to compute. If you do that, you see that I get 1 upon square of this and then you differentiate this. Finally, this is alpha e to the power of minus alpha x upon 1 plus e to the power of minus alpha x into 1 upon 1 plus e power minus alpha x. You can verify; this function is f of x and this function is alpha into 1 minus f of x. This f of x becomes f of x into 1 minus f of x and an alpha here. Although it is a nonlinear function, its computation is very simple because, it is in terms of f of x. That is the specialty of this activation function.

(Refer Slide Time: 06:45)



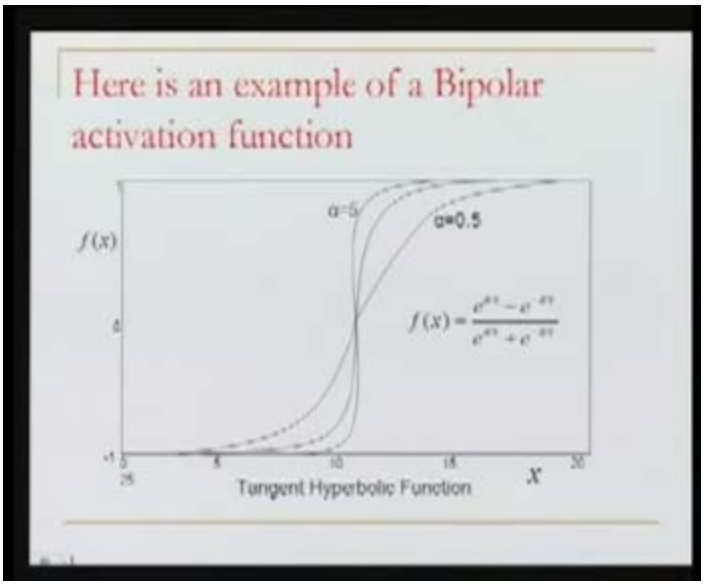
We derive for f of x is equal to 1 upon 1 plus e to the power of minus alpha x which normally we call as sigmoid function. For the sigmoid function, we showed that f of x equal to alpha f of x into 1 minus f of x. Because of this simple computation, this is one of the reasons also for making a choice of this activation function.

(Refer Slide Time: 06:51)



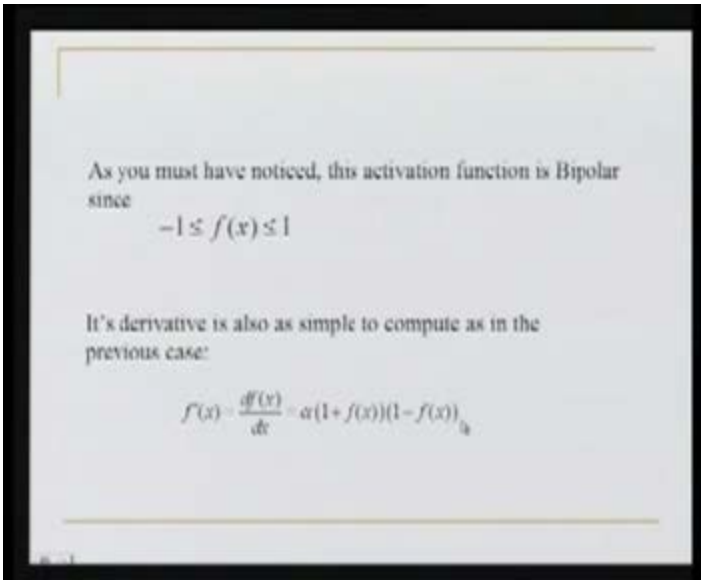
Here is another example of an activation function. The previous one was unipolar because the activation function it has a value, from 0 to 1 as you vary the input x, from minus infinity to plus infinity.

(Refer Slide Time: 07:39)



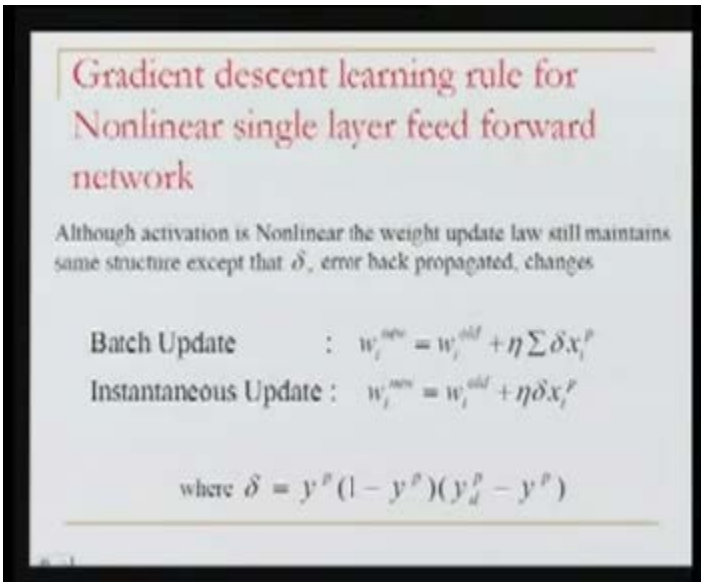
But now this activation function when you vary x from minus infinity to plus infinity you have value minus 1 to 1 and this is a tangent hyperbolic function; the tangent hyperbolic function e to the power of alpha x minus e to the power minus alpha x upon e to the power of alpha x plus e to the power of minus alpha x. Again alpha is there just to control the nonlinearity in the function.

(Refer Slide Time: 08:25)



The activation function has a limit from minus 1 to 1. So it is a bipolar activation function and as we derived earlier, you can derive and you can verify that the differentiation or derivative of this activation function f of x with respect to x is alpha into 1 plus f of x into 1 minus f of x. Again, it is a very simple computational form.

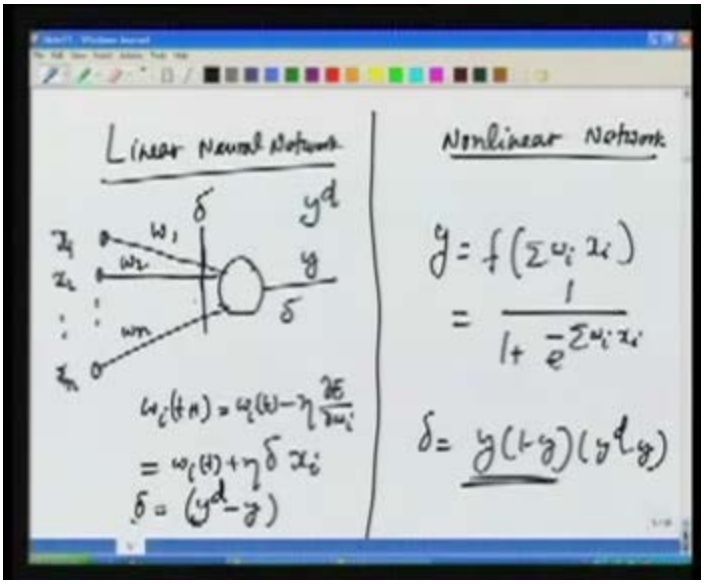
(Refer Slide Time: 09:12)



For this, if you use the sigmoid activation function that is, the previous one that is 1 upon 1 plus e to the power alpha x, then we can go back to the basic principle of gradient descent rule. If we apply the gradient descent rule for this nonlinear single layer feed forward network and if you apply the gradient descent rule, you will get batch update. This one is batch update and this is instantaneous update, where the delta is y p 1 minus y p y d p minus y p. These are all specific to specific pattern; p refers to a specific pattern. The structure of the update law remains same; what we had earlier, the same structure. The only difference is that back error propagated delta has become different.

I will go back now to make a little comparison between linear neural networks and non-linear neural networks in terms of the similarity and differences.

(Refer Slide Time: 10:26)



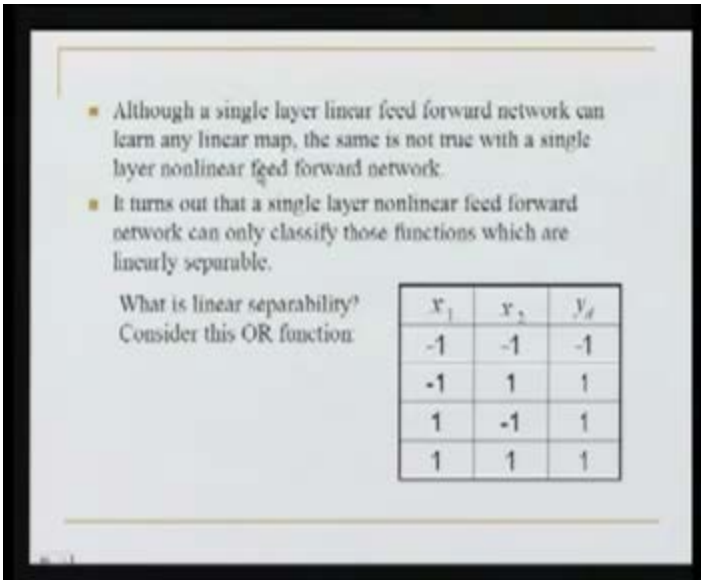
I have linear neural network on one side, here; here non-linear network. In a linear network, I have x_1 x_2 and x_n and then you have the computational unit; linear computational unit, w_1 w_2 w_n and here y. In case of linear this y is equal to sigma $w_i x_i$. In case of a non-linear, the same thing; it is a non-linear activation function. So, y becomes f of x. It the same as 1 upon 1 plus e to the power minus sigma $w_i x_i$. This is a non-linear network. What do we do? We apply the gradient descent rule. w_i t plus 1 equal to the w_i t minus eta into del E by del w_i. This is the gradient

descent rule. This is applicable for both linear neural network as well as a non-linear neural network.

We have already shown, for linear neural network that this becomes w_i t plus eta delta x_i, where delta is the error propagated y desired minus y. We are only considering instantaneous update because mostly in control system we will deal with the real time implementation. Real time implementation means instantaneous update, we cannot do a batch update. That is why …… we will only consider instantaneous update and for this delta is equal to y d minus y, when it is linear and you can compute the same thing here for non-linear network and you get delta is y 1 minus y y d minus y. Here we saw that delta for linear neural network is y d minus y and we computed the same delta for non-linear network; y 1 minus y, y d minus y. This is the extra term that comes because of differentiation of f. You can now appreciate why f has been selected as sigmoid activation function. This is one of the advantages of computation.

Let me take you through the total process of the update of weights in simple neural network; linear and non-linear. Here it is your input. Take the input pattern. Allow that pattern to be processed by the network. Network actuates y. For each input pattern, there is a given y d, desired output. Compute delta. This delta is y d minus y for linear network and for non-linear network, it is y 1 minus y y d minus y. So, that error has been transferred and then you look at here, in this side. This is your back error propagated, delta. This is your input pattern. Each weight if you look at is simply being updated based on its input, x_1 and what is the error being back propagated. You can look at here the update algorithm is w_i t plus eta delta x_i. For everything we can easily check, a very simple rule; the weight update is the original weight plus eta, the learning rate into the input and the back error propagated.

(Refer Slide Time: 15:15)

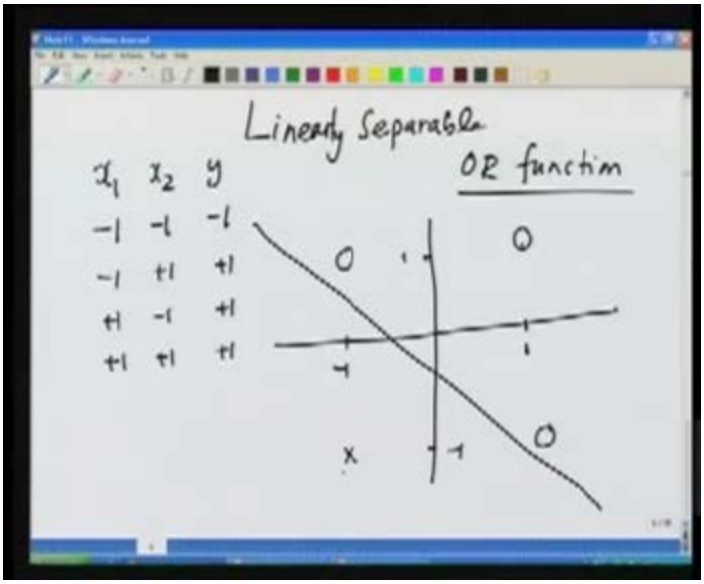


Although a single layer linear feed forward network can learn any linear map, the same is not true with a single layer non-linear feed forward network. We took last class and in today's class we showed that any linear dynamical system can be mapped or can be learned or can be identified using a single layer linear neural network. But this is not the case with non-linear dynamical function. We cannot identify or we cannot approximate any non-linear function using a single layer neural network. So that is the bad part, bad news. But it turns out that a single layer nonlinear feed forward network can only classify those functions which are linearly separable.

We saw the difference between weight updates in case of linear neural network and non-linear neural network when there is only single layer; simple neural networks. We also showed that any single layer linear neural network can approximate any linear dynamical system. If we could have mapped any nonlinear dynamical system using a simple non-linear neural network, probably we would have been happy, but this is not the case. It turns out that a single layer nonlinear neural network feed forward network can only map those nonlinear functions that are linearly separable. Those functions which are not linearly separable cannot be mapped or cannot be learnt using a single layer non-linear feed forward network. This is the bad news. So, what is this linear separability?

Let us consider a simple static function. All of you know an OR function, OR network. Any OR gate has two inputs, x_1 and x_2, output is y_d. These are the truth table. Input is minus 1 minus 1 output is minus 1 and in all other cases, the output is 1. That is the OR network.

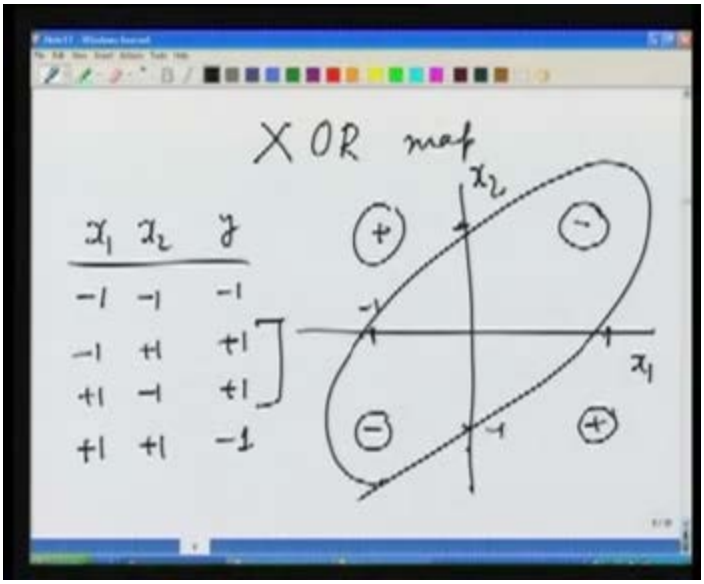
(Refer Slide Time: 18:51)



Linearly separable - we want to address this one. We have an OR function. All of you are already aware of the OR function. The truth table is this becomes minus 1, this becomes minus 1. In all other cases, this is plus 1. This is your truth table. Given specific input pattern, output is either minus 1 or plus 1. Now, let us look at graphical map of this function. (Refer Slide Time: 20:06) Minus 1 minus 1 is here. This is say 1. This is 1, minus 1 and minus 1. This is your point, where it is minus 1 and all other cases like minus 1 plus 1, you have plus and plus 1 plus 1, you have a plus. You have plus. You see that if I draw a line here, it is a linear line. It is able to separate a specific class from another class. For this class of input pattern output is 1 and for this class of input pattern, output is minus 1.

This particular function that is OR function is known as linearly separable function. This can be easily mapped using a non-linear single layer neural network; very simple. To conclude, a linearly separable function - a function is linearly separable if there exists a hyperbola that distinguishes between one class and another class. Thus linearly separable, but there are many cases, many functions which are not linearly separable. A very simple example is XOR function. Let us look at another example XOR map.

(Refer Slide Time: 21:36)

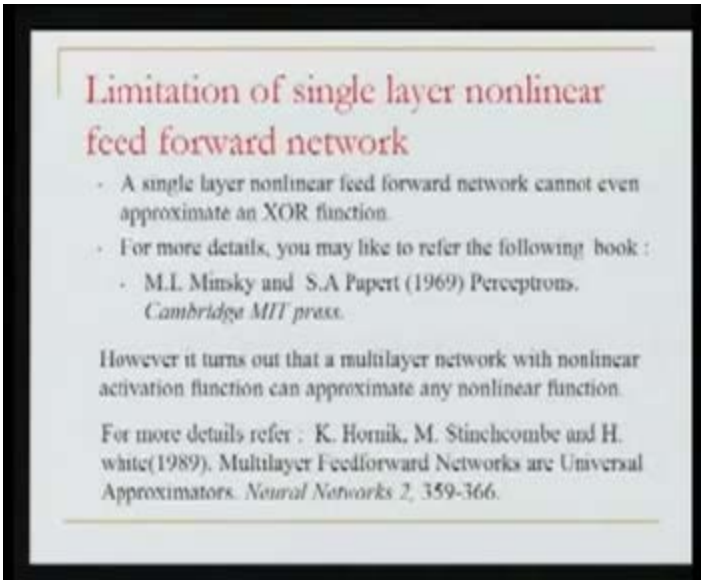


In this function, we have two inputs and one output and the truth table looks like this. This is your XOR map. For this case, you have plus 1 output and the other two cases you have minus 1 output. Let us look at the graphical map. When you have minus 1, This is your x_1, this is your x_2. So, minus 1 and minus 1, you have minus and when you have plus 1 and plus 1, you have again minus and other two cases you have here plus; that is plus 1 minus 1 and when you have minus 1 plus 1 you have also plus.

In this case, you cannot find a single line that will separate this class from this class. This class cannot be separated from this class using a single line. There is a function where the function is not linearly separable, but this is non-linearly separable. I can create a non-linear decision boundary that separates this class from these two classes. This class, the minus class is separated from plus class using a non-linear decision boundary. This is called the functions which are not linearly separable.

Let us summarize what we discussed now. Limitations of a single layer non-linear feed forward network is a single layer non-linear feed forward network cannot even approximate an XOR function; that is it can only approximate only those functions which are linearly separable.

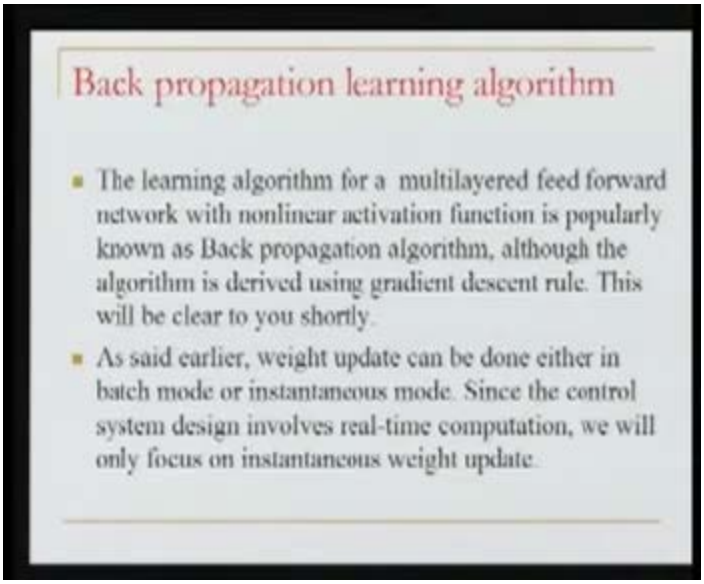
(Refer Slide Time: 24:16)



There are many complex functions. They are not linearly separable. You can refer to Minsky and Papert's book called "Perceptions" to know more about this linear separability and non-linear separability. This book is published by Cambridge MIT press, 1969. However, later researchers found that if we increase the complexity of the network in terms of the number of layers, but we only consider a single layer network; 2 layer, 3 layer, 4 layer that is multi-layer network, then such a feed forward network can always approximate any non-linear function.

One of the very key research contributions are by Hornik Stinchcombe and White published in 1989 in Neural Networks, Multilayer Feedforward Networks are universal approximators. The problem that we faced from single layer neural network for non-linear approximation can be mitigated or eliminated by increasing the layers to more than 1. It turns out that even a 2 layer network with 1 hidden layer, consisting of infinite neurons can also approximate a non-linear function.

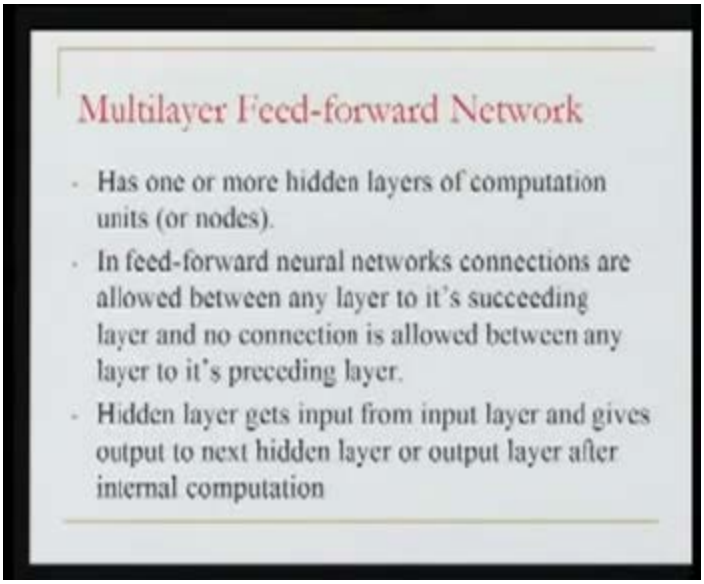
(Refer Slide Time: 26:05)



When this neural network becomes complex, can we write the learning algorithm, the region the way we wrote for single layer network? The answer is yes, little adjustments have to be done, but implementation-wise it is not difficult. The algorithm that was derived using gradient descent for nonlinear neural networks with nonlinear activation function is popularly known as back propagation learning algorithm, although the learning algorithm still is derived using gradient descent rule.

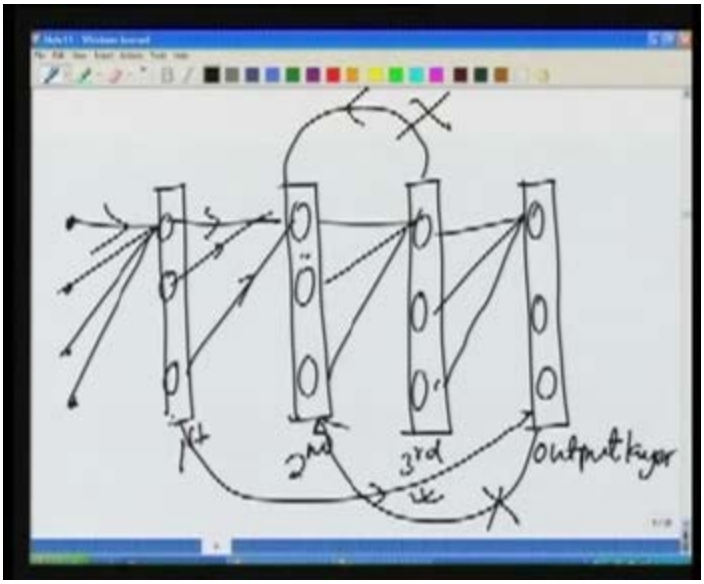
We will make this point very clear why this learning algorithm is known as back propagation. We gave some hints in the single layer network that computed error output is back propagated and based on that, the weights are being updated. Based on that information, the weight update formula takes a very simple shape, simple structure. Again, for a multi-layer network, is it very difficult to derive the learning algorithm? We will derive today the learning algorithm for a two layer network and in the next class we will show, for any number of layers, the hidden layers may be of any numbers, it can be in capital N, but the learning algorithm derivation is very simple. But all these learning algorithms that we will derive in this course will be instantaneous update rule. The reason being, again as I said, control systems are all real; they require real time implementation and we have to be considerate from that point of view. So, let us summarize what we are now going to do. This is the focus of today's class.

(Refer Slide Time: 28:10)



Multilayer feed forward network has more hidden layers and again, when I say feed forward network, the connections are all allowed only from any layer to its succeeding layer, but the connections are not allowed from any layer to its preceding layer. The example is you see here there are four layers. These are all inputs. First hidden layer, second hidden layer, third hidden layer and this is output layer. When we say the number of layers, we do not count the input layer as one of the layers. When I say two layered network, then I have only one hidden layer and next layer becomes output layer.

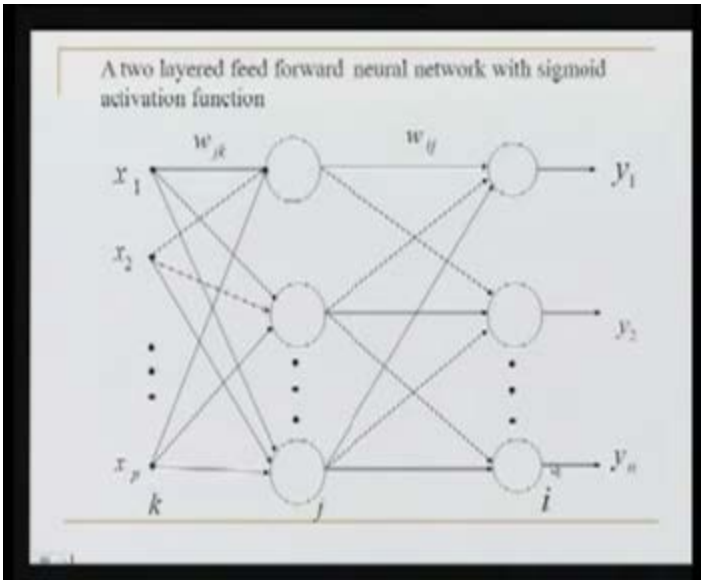
(Refer Slide Time: 30: 26)



This particular configuration means there are sub-units, sub-neurons here and this particular configuration, if I connect you will see why I say feed forward network, because I am able to connect any layer from its preceding layer. That means connections are allowed from the preceding layer to any layer, but I cannot allow the feedback connection. (Refer Slide Time: 30:54) This is called feedback connection; this is not allowed. This is allowed. From this layer, I can connect to this layer. This is allowed, but I cannot allow from this layer to connect to this layer. These are called feedback connections. They are not allowed and that is why this is known as feed forward network.

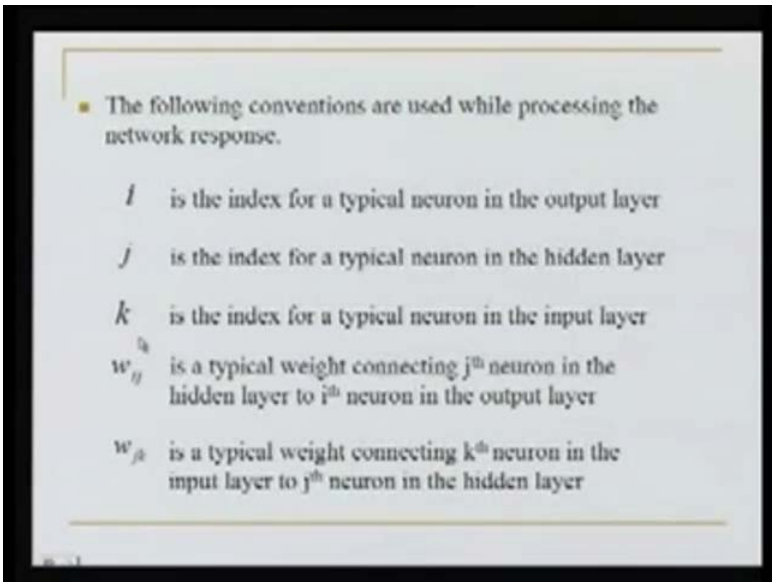
Today, we will derive a two-layered feed forward neural network with sigmoid activation function. We can very easily see that this is 1 layer; this is the only hidden layer and this is the only output layer; output layer is always only one.

(Refer Slide Time: 31:34)



What will we do? We have a certain convention that we will put while deriving a back propagation learning algorithm for this. The same simple principle; given training data, we allow the input to pass through the network, compute the error here, use the gradient descent rule and the back propagated error are used to modify the weights here that is between output layer and hidden layer and again another form of back propagated error here has to be used for modification of the weights between input layer and hidden layer. This is again the convention that we will use.

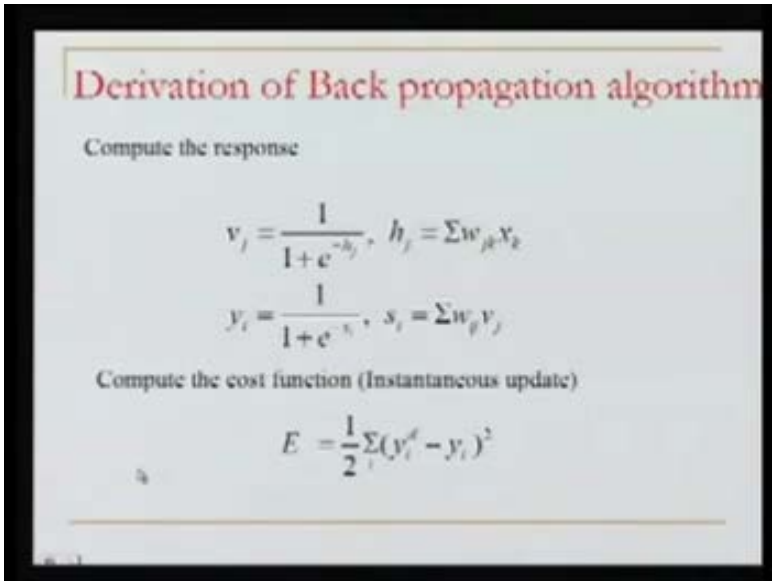
(Refer Slide Time: 32:20)



i is the index for a typical neuron in the output layer. This you can see here (Refer Slide Time: 32:30), i is the index for neurons that are used in this output layer, j is the index for neurons in the hidden layer and k is the index for the input patterns and the weights for a typical weight, that is between the hidden layer and the output layer is denoted or represented by w_{ij}. i is here and j refers to this particular layer and here a typical weight between input layer and output layer is represented by w_{jk}, where j refers to index for middle layer and k is the index for input layer. This is what we have said here. i, j and k are three different indexes for three different layers; input, hidden and output. This is the weight; typical weight representation between hidden layer and output layer and this is the typical weight representation for the weights between input layer and output layer.

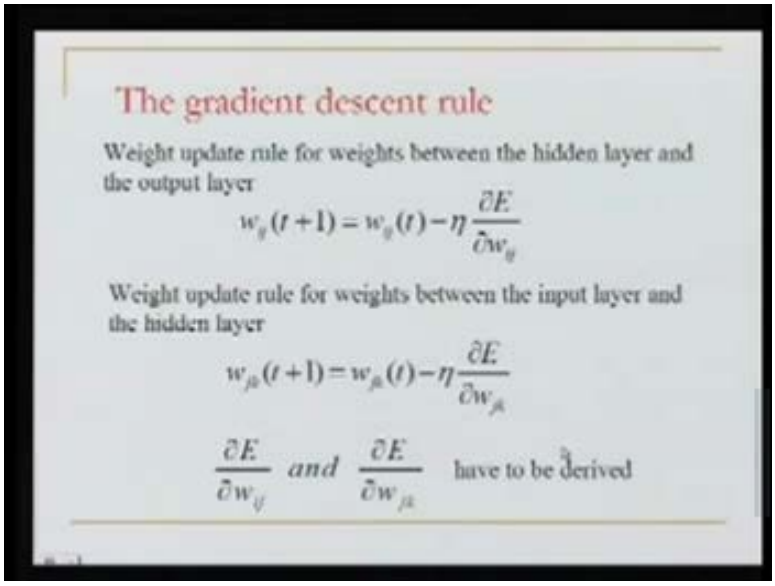
Now, let us see the derivation of the back propagation algorithm.

(Refer Slide Time: 34:03)



First what do we do? As I said, in the network you allow the input to pass through the network and compute the response y_1 y_2 y_n. To compute y_1 y_2 y_n, we assume that the hidden units had output also u_1 u_2 u_3 and these output are first computed; the outputs of the hidden units are first computed and using the output of the hidden units, the output of the output layer are computed. Just for clarity, again I say, all these neurons are sigmoidally activated. They use sigmoid activation function. We can say that v_j is 1 upon 1 plus e power minus h_j, where h_j is the total input reaching the jth neuron of hidden layer. Similarly output of the ith neuron in the output layer is 1 upon 1 plus e to the power minus s_i, where s_i is equal to sigma into $w_{ij}v_j$. s_i is the total inputs reaching the ith neuron in the output layer. Here we are computing the cost function for instantaneous update which is this value equal to half summation y_i d minus y_i squared.

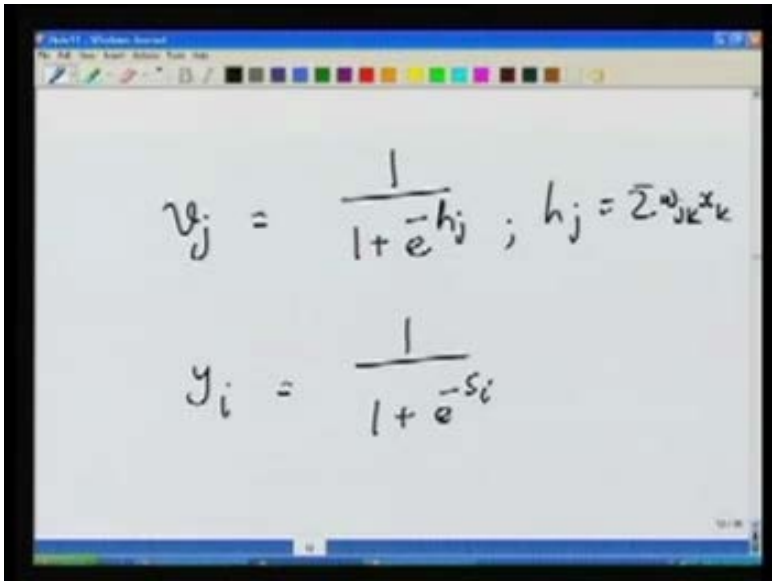
(Refer Slide Time: 35:48)



So, the gradient descent rule is the same. The weight update rule for weights between hidden layer and output layer, we will have this particular ….. That is the principle of gradient descent w_{ij} into t plus 1 equal to w_{ij} t minus eta del E divided by del w_{ij}. Weight update rule for weights between the input layer and the hidden layer is w_{jk} t plus 1 equal to w_{jk} t minus eta del E divided by del w_{jk}.

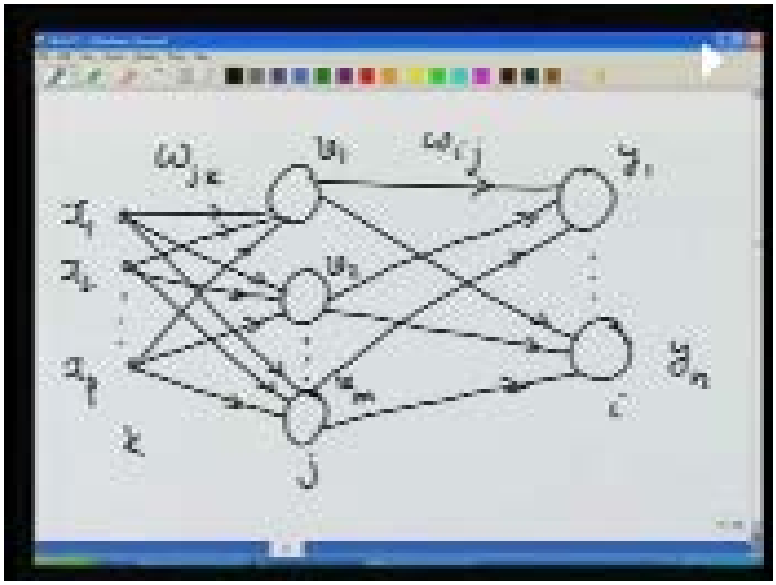
How do you find now del E by del W_{ij} and del E by del W_{jk}? This is what we have to find out. Here we showed this diagram. Again, I drew in the black board for clarity, because I would like to derive the whole thing in black board. You have input x_1, x_2 up to x_p and output y_1 to y_n, n different computational units and you have hidden layer with m hidden neurons, whose outputs are v_1, v_2, v_n and as I said, w_{ij} is the typical weight connecting ith neuron in the output layer with jth neuron in the hidden layer. Similarly, w_{jk} is the typical weight between hidden layer and input layer. What will we do is in the first phase we will allow this input to pass through the network and we compute what is y_1. So, let us compute that.

(Refer Slide Time: 37:36)



What we are doing is we are computing the output of a hidden neuron, v_j. v_j, the output of the jth neuron of the hidden layer is 1 upon 1 plus e to the power minus h_j where h_j is the total input reaching the jth neuron and h_j is w_{jk} x_k.

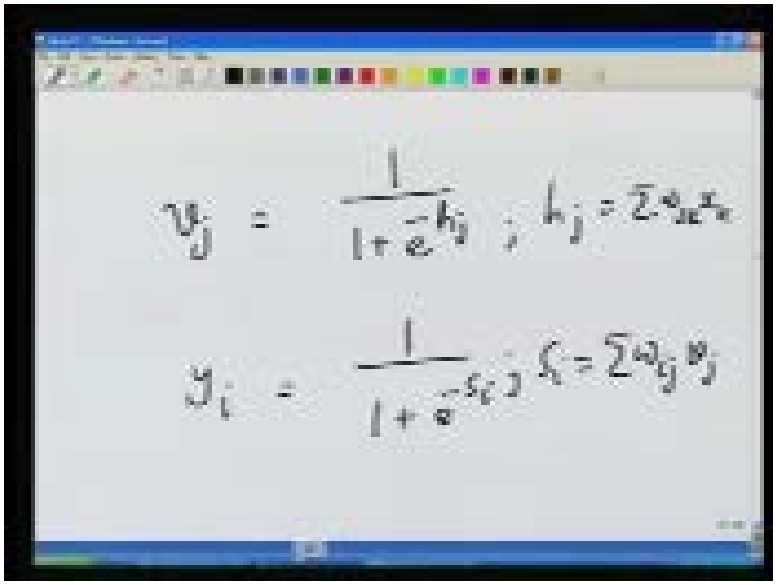
(Refer Slide Time: 37:57)



If we go back, you see here v_j. Say for example, I want to compute v_1. What do I do? I compute what is h_1? h_1 is sigma w_1k into x_k. That is w_{11} into x_1 plus w_{12} into x_2 so on plus w_{1p} into x_p.

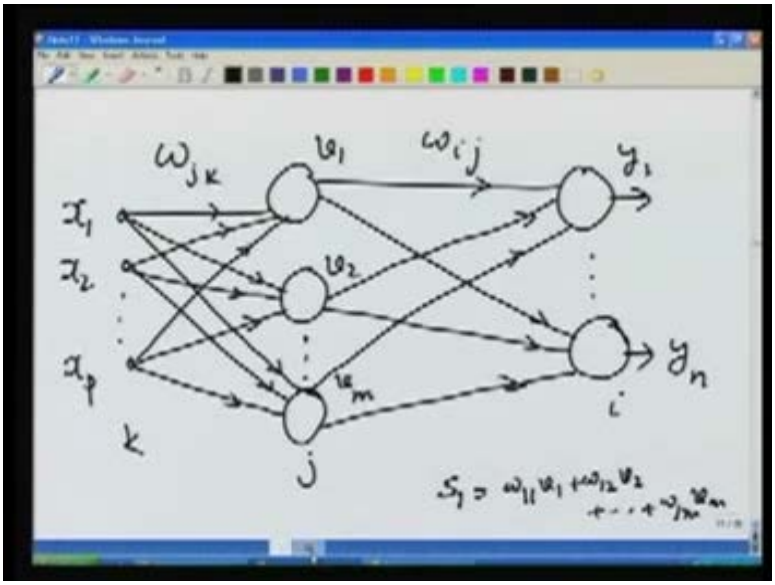
This way we compute what is v_1, v_2 and v_m and after we computed v_1, v_2, v_m, we will now compute what is y_1 up to y_n?

(Refer Slide Time: 38:55)



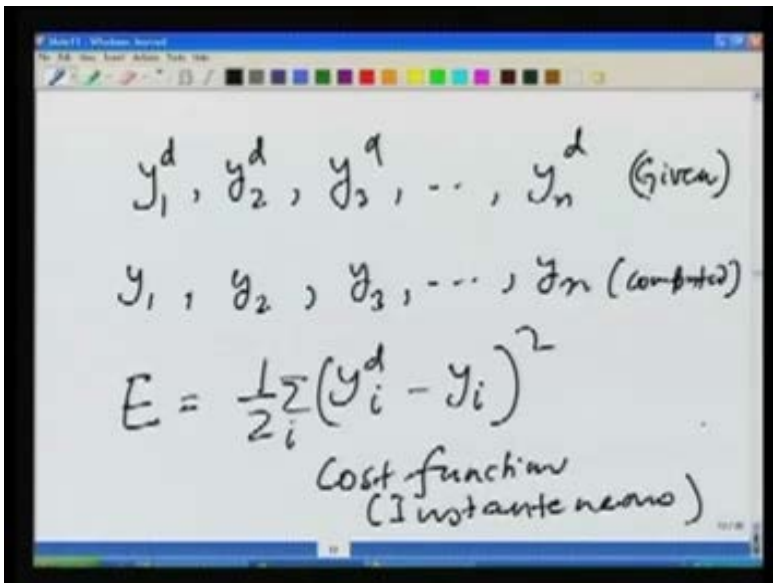
This computation is y_i is 1 upon 1 plus e to the power minus s_i where s_i is sigma w_{ij} in to v_j. This you can also verify in the previous one.

(Refer slide time: 40:01)



For example, I want to compute y_1. First I will compute what s_1 is and you can say s_1 is w_{11} v_1 plus w_{12} v_2 and so on plus w_{1m} and v_m. Once you compute s_1, the next .... is y_1. y_1 is 1 upon 1 plus e power minus s_i. This is a sigmoidal activation function. We are done with how to compute y_1 to y_n given x_1, x_2 and x_p using forward propagation.

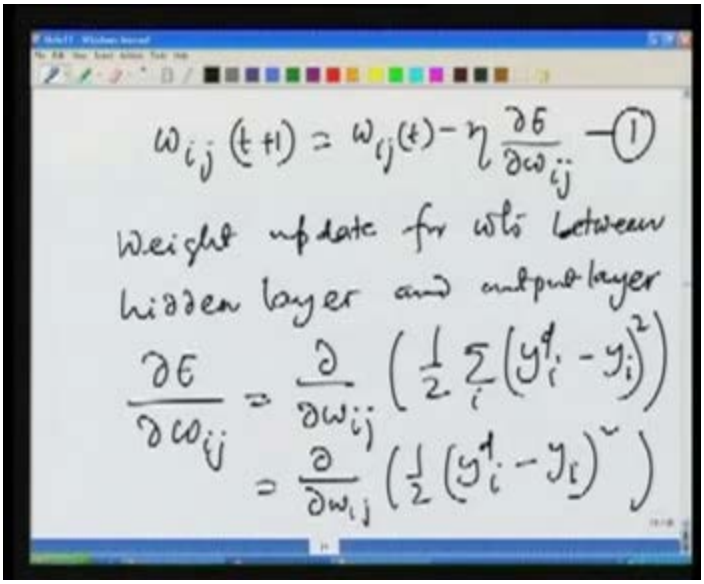
(Refer Slide Time: 41:32)



We go to computing the error, error at the output. I am given actually y_1 d, y_2 d, y_3 d and y_n d. This is given. I compute what is y_1, y_2, y_3 and y_n? This is computed. The network computes y_1, y_2, y_3 and y_n given y_1 d, y_2 d, y_3 d and y_n d. So, I compute the cost function. The instantaneous cost function is half y d i minus y_i square sigma over i. I subtract from y desired y, square it and add all the output units in the output layer. That is how I compute the cost function. This is called instantaneous cost function and this instantaneous function is computed for a specific pattern, for a given pattern I compute this cost function.

Now, I go to the next. Once I compute what is E, I have to compute for the weight, a typical weight between hidden layer and output layer w_{ij}, the weight whatever was in previous iteration whatever is w_{ij} that has to be updated by adding a gradient term.

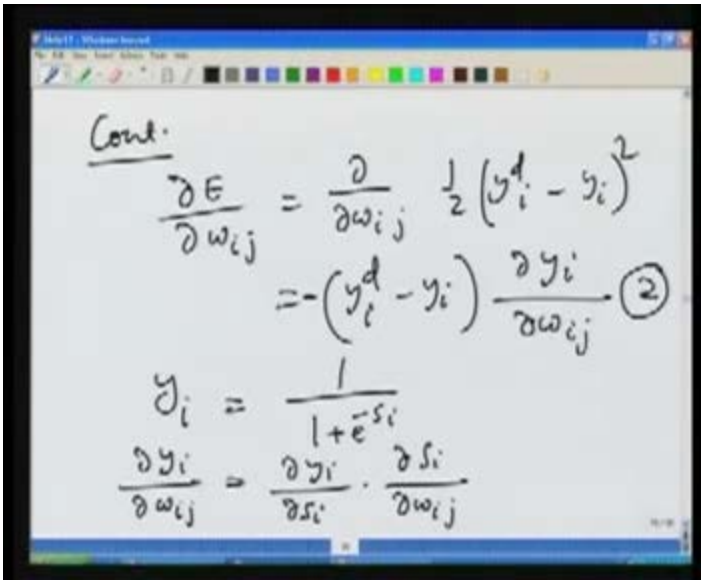
(Refer Slide Time: 42:50)



This is the weight update for weights between hidden layer and output layer. All that you have to do is simple derivative principle. How to differentiate given y and you know the function structure; so, you simply differentiate. All that we have to do is that del E upon del w_{ij}. If I can compute I just have to put it there in this equation. So this is equation 1. I have to put that I differentiate del by del w_{ij} and here I have half sigma over i y d i minus y_i whole square. You can see that only the error contribution due to ith neuron is a function of w_{ij}.

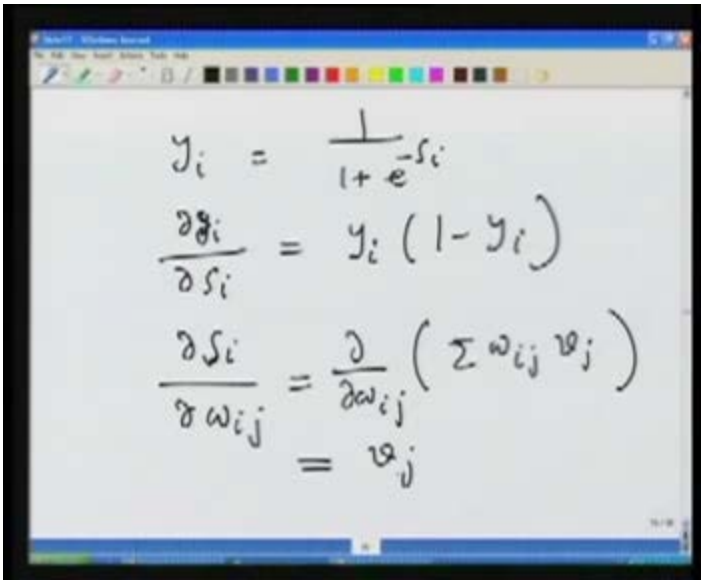
Hopefully, you are very clear. If I am considering this particular unit this is my typical weight, which is w_{11} and I want to update $w_{11.}$ Then you should be very clear that only the error computed here at y_1 is a function of $w_{11.}$ You can very well see that y_n is not a function of $w_{11.}$ Once you are very clear about that I can write this one as del by del w_{ij} half y d i minus y_i whole square. Other terms are not relevant.

(Refer Slide Time: 44:59)



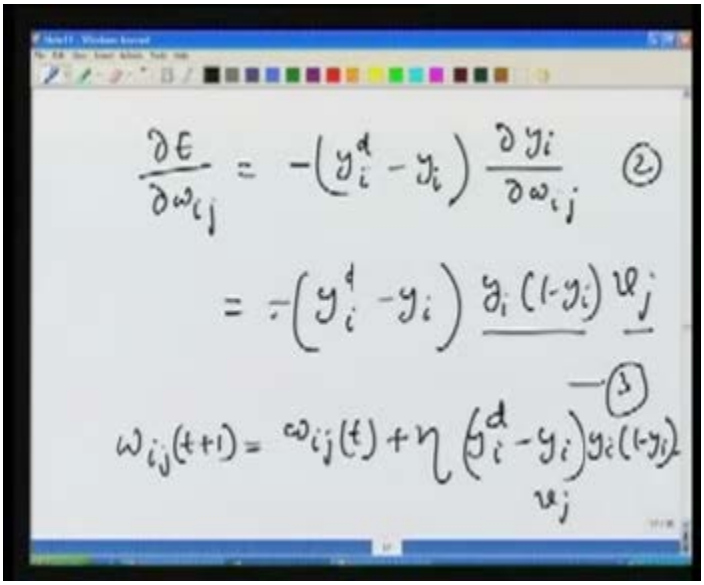
We go again, we continue; del E by del w_{ij}, we found out to be del by del w_{ij} half y d i minus y_i square. You can check it; y i d minus y_i into, here minus, del y_i by del w_{ij}. Hopefully, you are now clear with this expression. This is, I can say, equation number 2. From equation number 2, we have to find out what is del y_i by del w_{ij}? What is y_i? y_i is 1 upon 1 plus e to the power of minus s_i. So, del y_i by del w_{ij} has to be written like this; del y_i upon del s_i into del s_i by del w_{ij}.

(Refer Slide Time: 46:13)



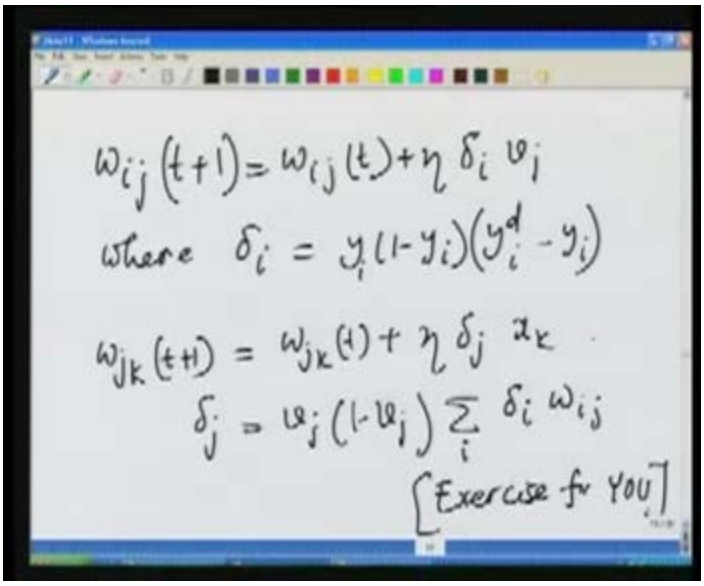
We go to the next step. If y_i is 1 upon 1 plus e to the power minus s_i, we have already discussed this kind of activation function, if I differentiate s_i then I find the answer is y_i 1 minus y_i. This we have already said. The rest is left is del s_i by del w_{ij}. To find out del s_i by del w_{ij}, I must write down del by del w_{ij} and s_i is sigma w_{ij} into v_j. This is simply v_j because this is simply summation of terms. So, with respect to w_{ij}, only v_j comes out. If we want to differentiate with respect to w_{ij} v_j comes out.

(Refer Slide Time: 47:33)



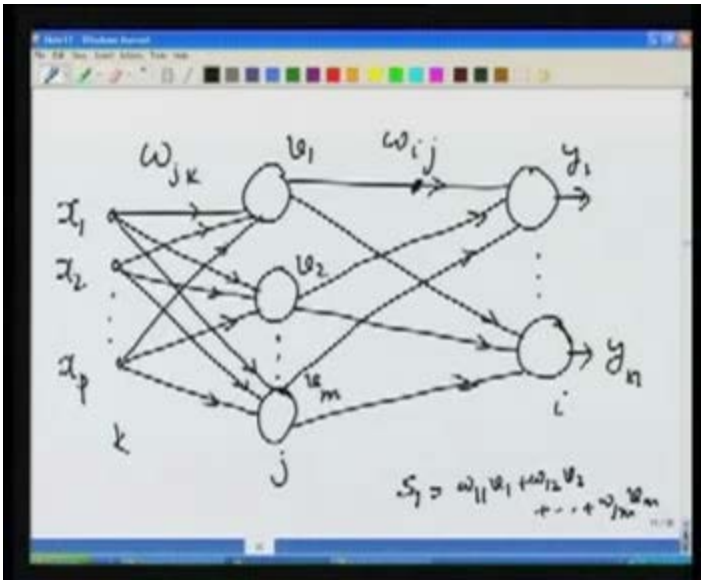
We have finally reached this solution that is del E by del w_{ij} is minus y d i minus y_i into del y_i upon del w_{ij} which was equation number 2. That reduces to minus y d i minus y_i into y_i 1 minus y_i into v_j. This is the final expression; third. This term comes from del y_i upon del s_i and this term comes from del s_i upon del w_{ij}. What was our earlier update equation? w_{ij} t plus 1 equal to w_{ij} t plus, this was actually minus, I am making plus because I am getting here minus eta del E upon del w_{ij}. That was minus; here the term is minus so, this becomes plus eta y i d minus y_i into y_i 1 minus y_i into v_j. This is your final equation for the weight update law for weights between hidden layer and output layer.

(Refer Slide Time: 49:24)



Finally, I write down the weight update equation for the weights between the hidden layer and output layer is eta delta i v_j where $delta_i$ is y_i 1 minus y_i ……. This should not be surprising to you, because for single layer neural network also for non-linear neural network we found delta to be of this form and the input is v_j. I will not now derive the next update equation that is required for the weights between hidden layer and input layer which is w_{jk} t plus 1, which also has this similar formula that is w_{jk} t. You can take just little pain to find out. This is $delta_j$ x_k. You say that it is a very nice formula where $delta_j$ is v_j 1 minus v_j sigma $delta_i$ w_{ij} over i. I give this as an exercise for you to derive this weight update equation.

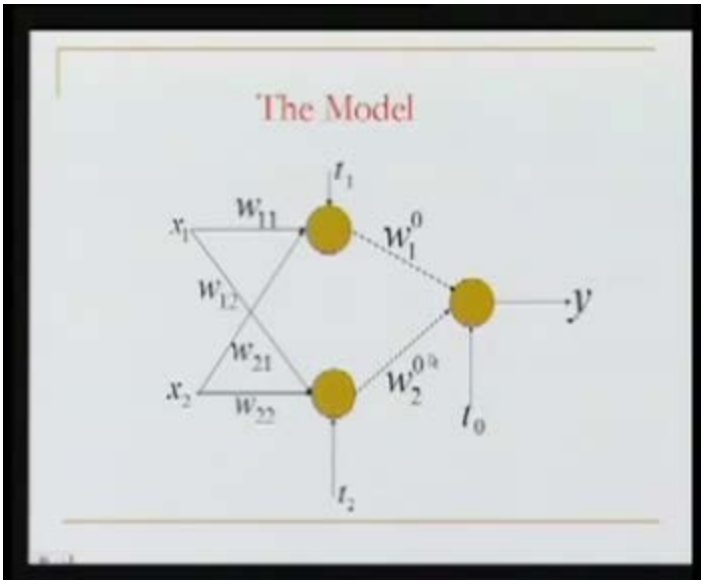
(Refer Slide Time: 51:30)



We are done with the derivation of the back propagation learning algorithm. I will just explain in the figure; we pass the information from here to here. We have a general weight update algorithm for these weights here and another general weight update algorithm for the layers here. So, that is the summary of the weight update algorithm.

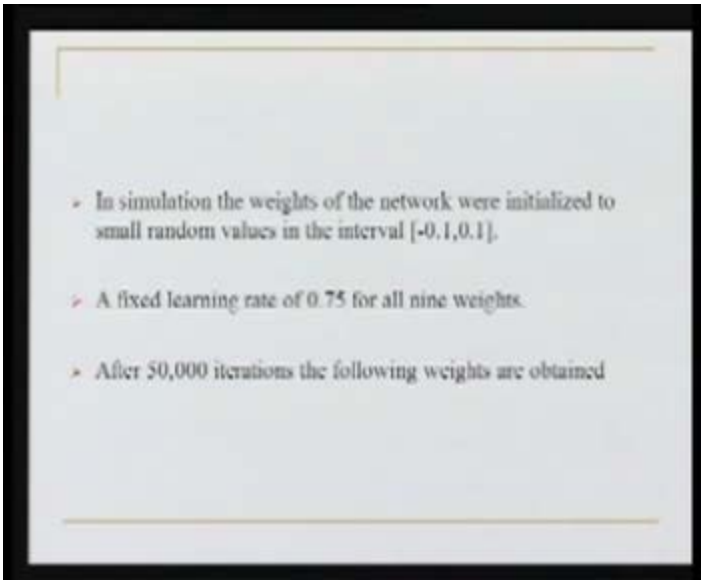
What you learnt today is the gradient descend algorithm. The derivation of this algorithm for multilayered neural network, we will apply that derivation for learning the XOR map which you could not solve using single layer feed forward network using non-linear activation function. There as usual this is our cost function and this is our model of XOR network.

(Refer Slide Time: 52:39)



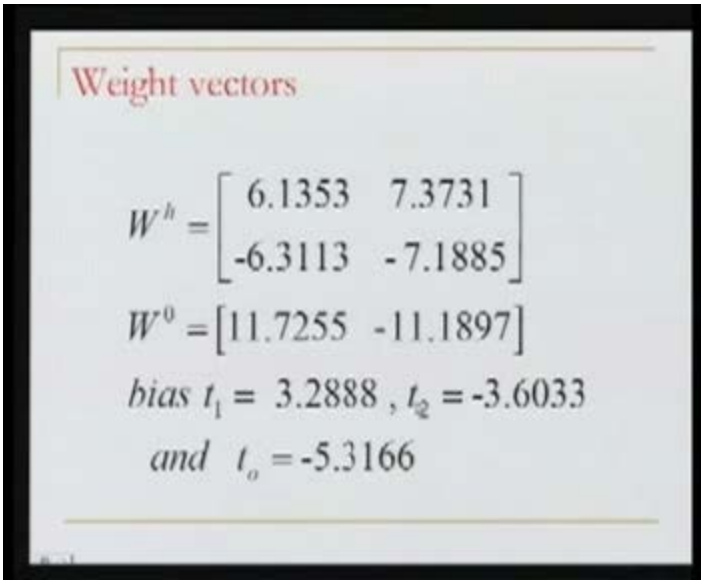
We have two inputs x_1 and x_2. These are our weights and we have put two bias units. This is the bias weight. That is one, I have a fixed input 1 and weight is t_1, here is t_2. You can also say this is If you consider this 1 to be here and another input, external input then you can also easily say, this is w_{13} and this is w_{23}. This is w_{13}. It has been represented by t_1 here and w_{23} is represented by t_2 here and these weights just for clarity, we have kept w 1 0 and w 2 0 and the bias weight is kept as t_0. That is input is 1 and the weight is t_0 and the output is y. Now, you update the weight. This is t_0, w 2 0 and w 1 0 using the weight update algorithm for w_{ij} and t_2, t_1, w_{11}, w_{12}, w_{22}, you update using the weight update algorithm for w_{jk}.

(Refer Slide Time: 54:01)



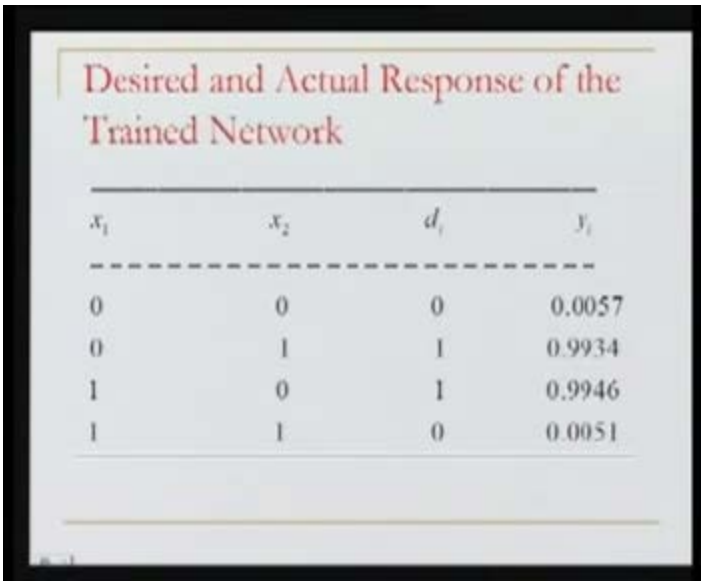
We do that. To start with all these weights you initialize between minus 0.1 to 0.1 and eta has been taken as 0.75 and let us take the number of iterations as 50,000. What is iteration? I take all these patterns sequentially. I have four patterns and 50,000 means I have taken these four patterns again and again until 50,000 iterations are over. If you do that, you see the weight vector that is obtained between hidden layer and input layer is this one.

(Refer Slide Time: 54:58)



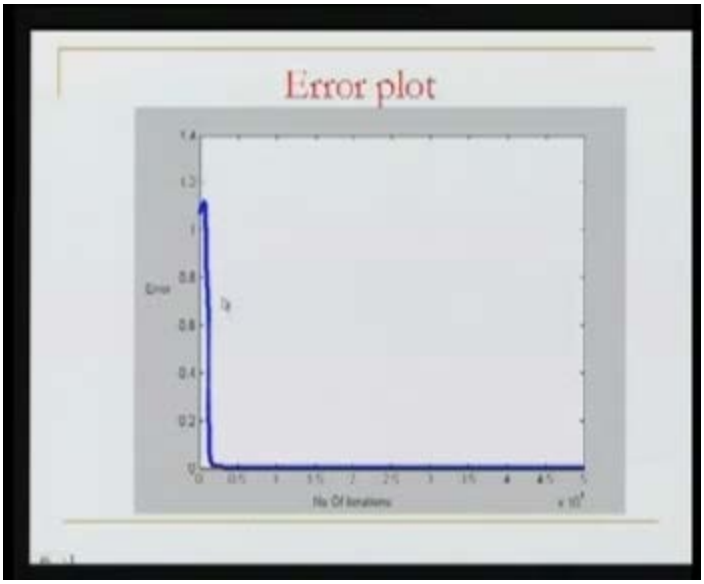
This is the output layer, the weight between the output layer and the hidden layer. These are the bias weights t_1 and t_2. Again, you consider this is for hidden layer and input layer and this is for output layer, the bias.

(Refer Slide Time: 55:20)



After training if you give this input, this is x_1 and this is x_2. This is my desired output and actual output is very close to desired output. You can say this is almost exact.

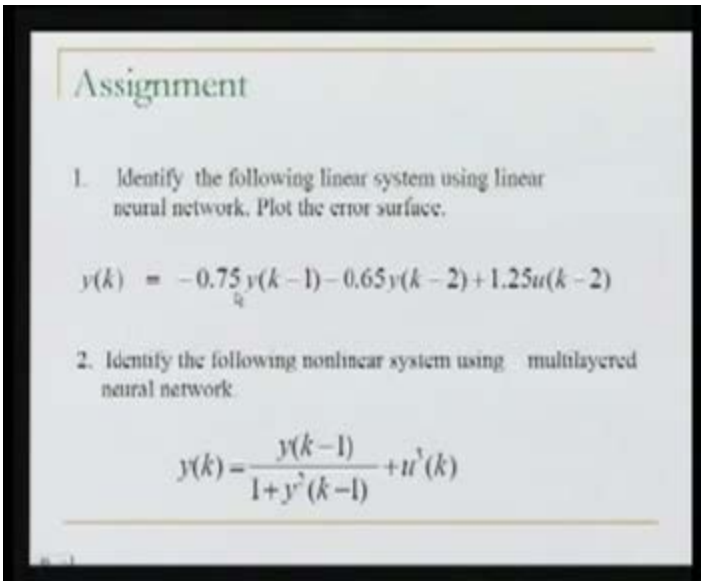
(Refer Slide Time: 55:38)



This is your error plot. You can easily see that although we have taken 50,000 actually within 5000, the training is over. It is not necessary actually to go up to this and in subsequent lecture we will say why we should not train further after training is over because after 5000 literally there is no training and this causes the problem of over generalization and that will be discussed in the next class.

Now here is assignment for you. The first one; please note down this linear dynamical system where the coefficients are minus 0.75.

(Refer Slide Time: 56:27)



This is second order system minus 0.65 and here 1.25 your law that This is w_1, w_2 and w_3. Assume w_1, w_2, w_3 to be very small random number. Use the training data using this model where u is a random number uniformly generated between 0 to 1 and apply this input data, use this input data to train linear neural network and you should finally get the answer; w_1 should be minus 0.75, w_2 should be minus 0.65 and w_3 should be 1.25. This answer you should get.

The second one maybe little difficult for you; this you have to use a multilayered neural network, feed forward network. I suggest you use around 10 hidden neurons. You have one output; the output layer has only 1 neuron, input also has only 1 u k. Take only 1 neuron in the input layer. So, the input layer has only one input. You can include one bias input. You generate the data for this, using this actual model taking u k again randomly generated number from 0 to 1 and then

normalize this values y k and u k between 0 to 1 because your neural network has an activation function whose output is between 0 to 1.

You have to normalize output also between 0 to 1 and after that, you train. After training just like you could correlate in case of linear dynamical system, you cannot correlate, because your neural network becomes a black box; we will discuss this more but what you can verify is you give new data to this actual model, new input data and for that new input data find out what is the output. Giving this new input data to your trained neural network, you should be able to get what is the desired output. Good bye. We will meet again in next class.