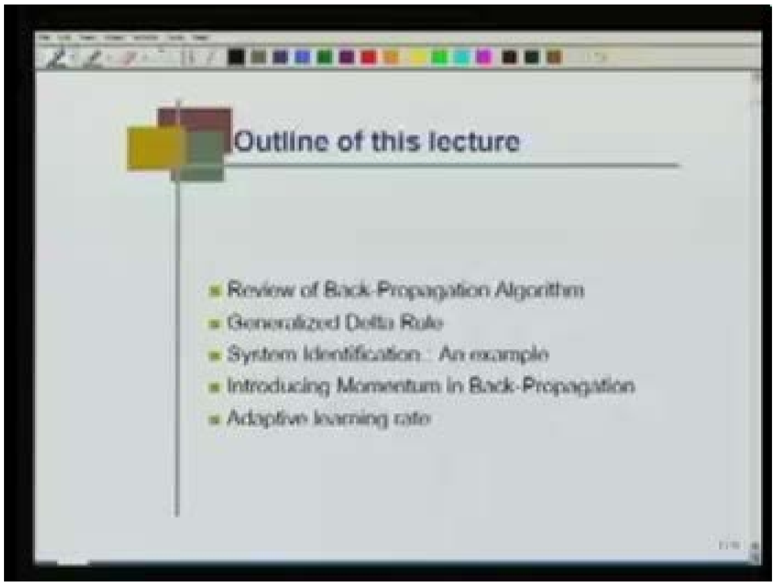
Intelligent Systems and Control

Prof. Laxmidhar Behera
Department of Electrical Engineering
Indian Institute of Technology, Kanpur

Module - 1 Lecture - 3
Back Propagation Algorithm: Revisited

This is the third lecture of module 1 – Neural Networks. We have already discussed the linear neural network and system identification in linear neural network; then, we talked about the feed forward neural network, that is, the multilayer network, which has the capability of approximating any nonlinear function. We learnt in the second lecture how to derive the back propagation algorithm for the feed forward neural network that can approximate any nonlinear function.

(Refer Slide Time: 01:11)

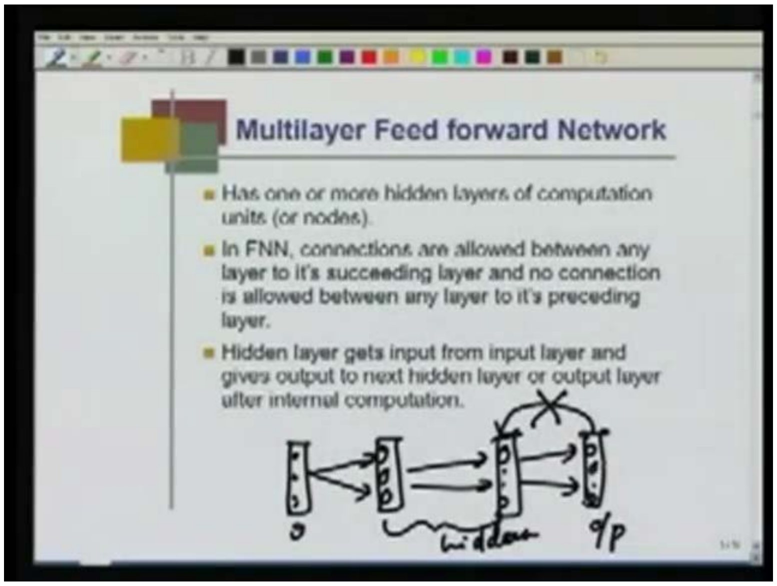


Outline of this lecture

- Review of Back-Propagation Algorithm
- Generalized Delta Rule
- System Identification : An example
- Introducing Momentum in Back-Propagation
- Adaptive learning rate

Today, in this third lecture, we will again review the back propagation algorithm that we derived in the last lecture; then, we generalize the delta rule and we will understand this concept today. Then, system identification using this back propagation algorithm; then two different variations in back propagation algorithm, that is, adding a momentum and adaptive learning rate. Today, we will just have a very heuristic version of the adaptive learning rate. Probably after three or four lectures, you will have a detailed analysis on how to comprehensively design adaptive learning rate for back propagation network, the reason being that this particular analysis would need the concepts of stability for nonlinear systems, that is, Lyapunov function and Lyapunov function-based stability theory. These notions have to be reviewed, before we can talk about a very comprehensive method of computing adaptive learning rate for back propagation network.

(Refer Slide Time: 02:27)

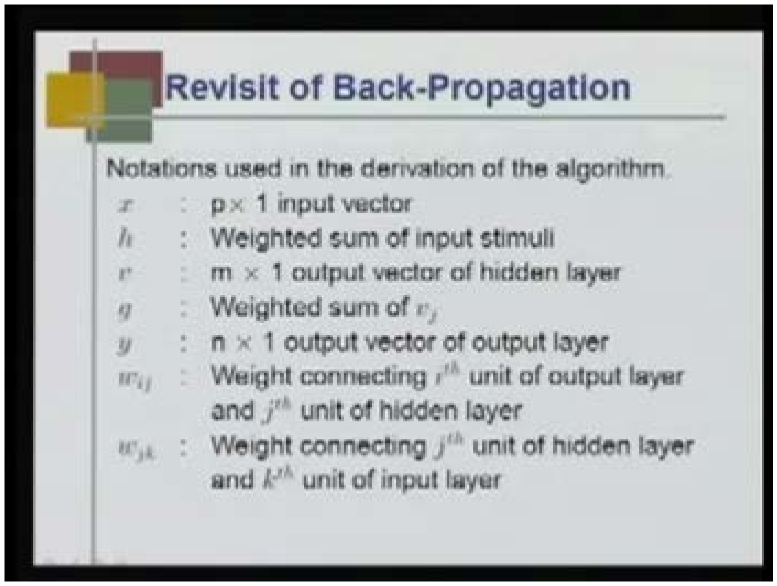


Multilayer Feed forward Network

- Has one or more hidden layers of computation units (or nodes).
- In FNN, connections are allowed between any layer to it's succeeding layer and no connection is allowed between any layer to it's preceding layer.
- Hidden layer gets input from input layer and gives output to next hidden layer or output layer after internal computation.

We have already discussed about multilayer feed forward network. We said that you have an input layer and then you have many layers. These layers have many neurons. These are all input signals coming here (Refer Slide Time: 02:52) and they are fanned out. This is your output layer and these all are hidden layers. This is called the multilayer feed forward network. It has more hidden layers of computation units, that is, these hidden layers can be more than one. You can have one, two, three, four – as many as we want. FNN stands for feed forward neural network. In this feed forward neural network, the connections are allowed from one layer to the succeeding layer in the forward direction and not in the backward direction. I cannot have a connection like this. This is not allowed, this connection is not allowed – from this layer to this layer (Refer Slide Time: 03:56), but it is allowed from this layer to this layer. The hidden layer gets input from input layer and gives output to the next hidden layer or output layer after internal computation. This is the basic structure of feed forward network.

(Refer Slide Time: 04:14)



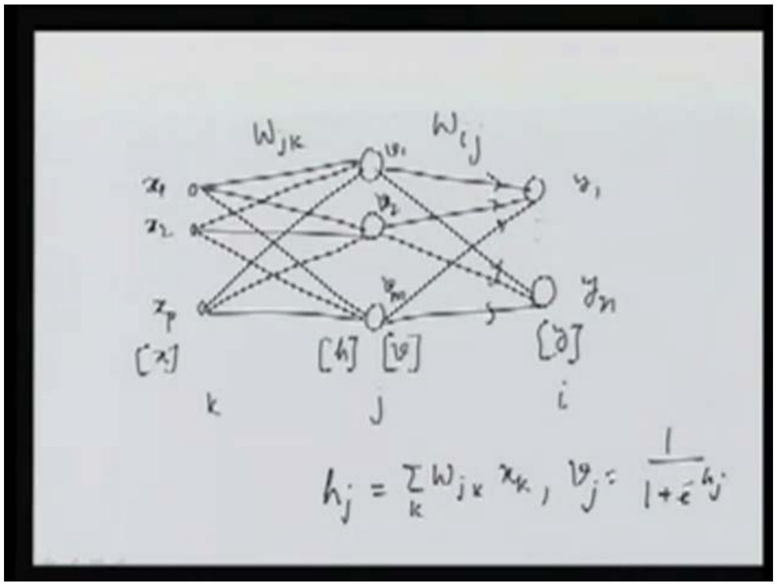
Revisit of Back-Propagation

Notations used in the derivation of the algorithm.

x	:	p× 1 input vector
h	:	Weighted sum of input stimuli
v	:	m × 1 output vector of hidden layer
g	:	Weighted sum of v_j
y	:	n × 1 output vector of output layer
w_{ij}	:	Weight connecting i^{th} unit of output layer and j^{th} unit of hidden layer
w_{jk}	:	Weight connecting j^{th} unit of hidden layer and k^{th} unit of input layer

This is the notation we have used in the last class about the back propagation network. x is the input, v is the hidden node output, y is the actual output, w_{ij} is the weight connection between i th unit of output layer and j th unit of hidden layer, w_{jk} is weight connecting j th unit of hidden layer and k th unit of input layer.

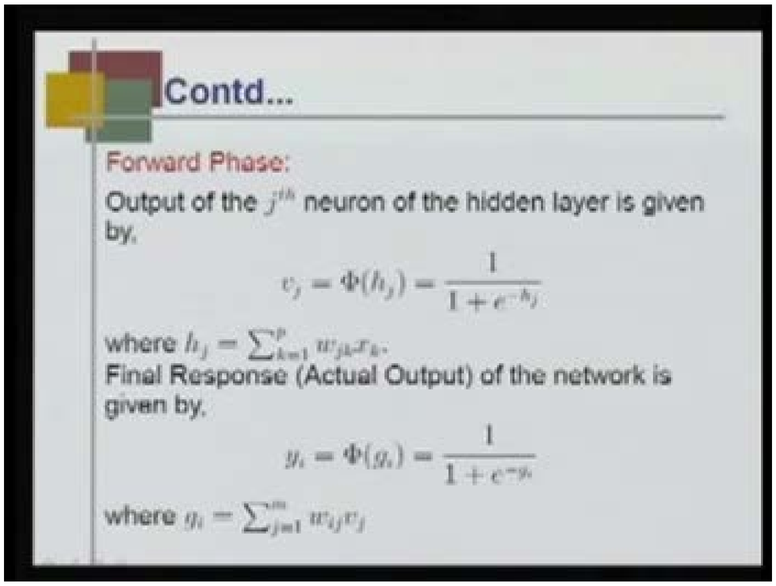
(Refer Slide Time: 04:44)



$$h_j = \sum_k W_{jk} x_k, \quad v_j = \frac{1}{1+e^{-h_j}}$$

You can easily see that x is the input, which is a p-dimensional vector. The summation of weights into the input signals are all summed here and that sum is h. If I say h_j, it is simply sigma over k w_{jk} x_k – this is your h_j. After you go through this sigmoidal activation, that becomes your v, so v_j is simply 1 upon 1 plus e to the power of minus h_j – this is after sigmoidal activation. Again as usual, output y is an n-dimensional output. y_i is…. You add all v_1. We have shown this here.

(Refer Slide Time: 05:53)



Contd...

Forward Phase:

Output of the j^{th} neuron of the hidden layer is given by,

$$v_j = \Phi(h_j) = \frac{1}{1+e^{-h_j}}$$

where $h_j = \sum_{k=1}^{p} w_{jk} x_k$.

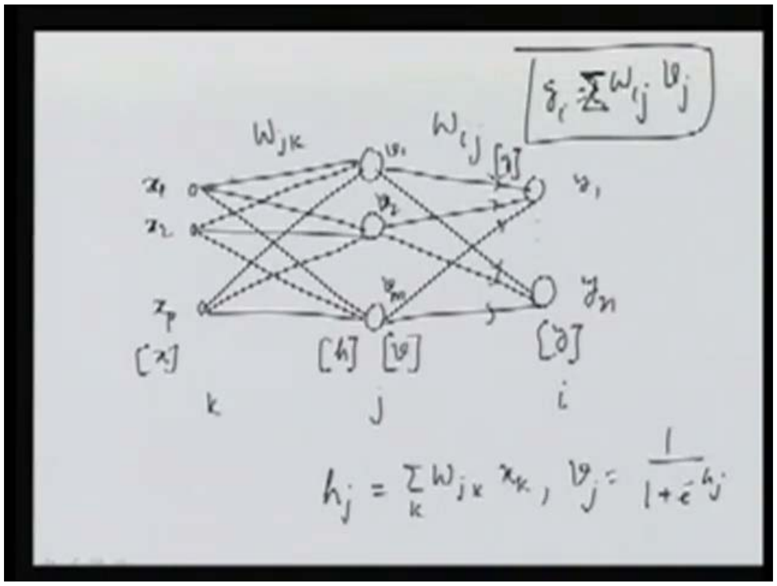
Final Response (Actual Output) of the network is given by,

$$y_i = \Phi(g_i) = \frac{1}{1+e^{-g_i}}$$

where $g_i = \sum_{j=1}^{m} w_{ij} v_j$

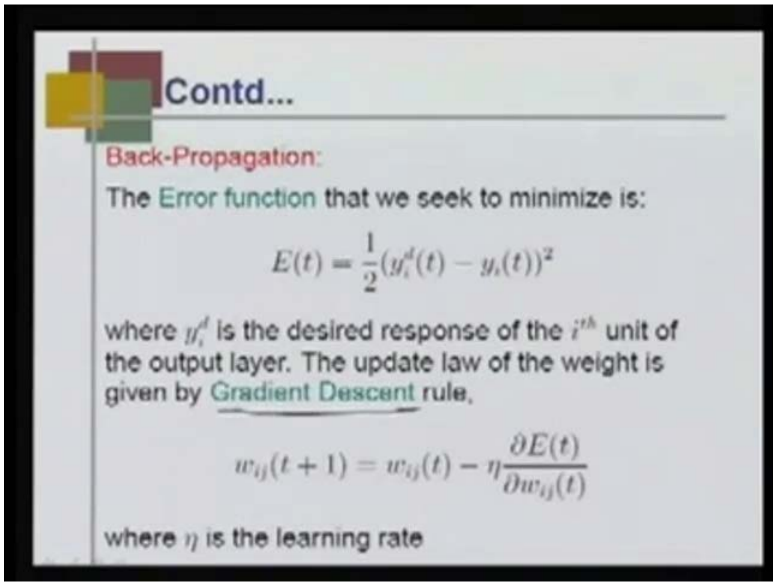
v_j is 1 upon 1 plus e to the power of minus h_j, where h_j is w_{jk} x_k and k equal to 1 to p. Similarly, final response y_i is phi g_i, which is 1 upon 1 plus e to the power of minus g_i, where g_i is w_{ij} v_j.

(Refer Slide Time: 06:16)



If I go back here, I write here g and that means g_i is simply sigma w_{ij} into v_j. This is sigma (Refer Slide Time: 06:31). What you are seeing is that w_{ij} is the connection weights between the output layer and the hidden layer and w_{jk} is the typical weight between the hidden layer and the input layer. This is the notation we used in the last class. This is called forward phase. In the forward phase, we computed the output of the hidden unit and output of the output unit.

(Refer Slide Time: 07:02)



Contd...

Back-Propagation:

The Error function that we seek to minimize is:

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

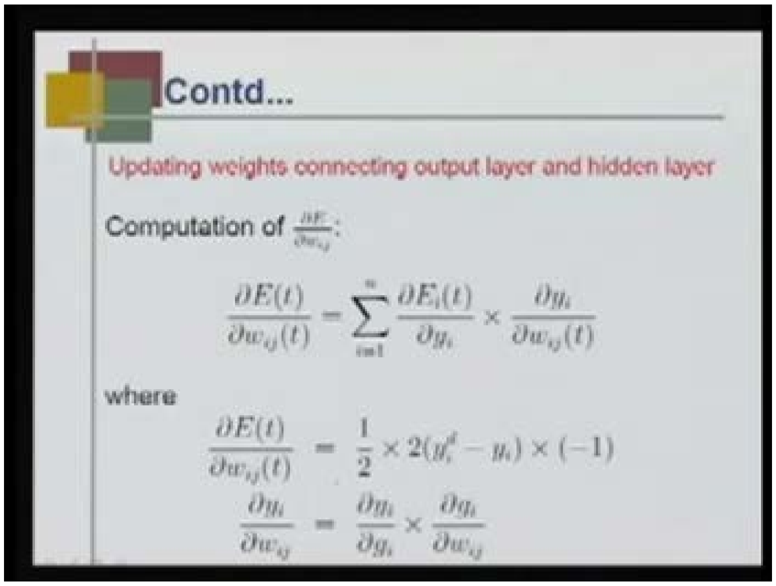
where y_i^d is the desired response of the i^{th} unit of the output layer. The update law of the weight is given by Gradient Descent rule.

$$w_{ij}(t+1) = w_{ij}(t) - \eta \frac{\partial E(t)}{\partial w_{ij}(t)}$$

where η is the learning rate

Then, back propagation. Given the input output patterns, given input x, I know what the desired target – y d is. The network has a response y_i and so, I compute a cost function E of t, which is a quadratic cost function. I use the gradient descent rule, which is of the nature w_{ij} into t plus 1 is w_{ij} of t minus eta into dow E of t upon dow w_{ij} of t. We have already discussed a lot about gradient descent; we apply this gradient descent to compute the error back propagation.

(Refer Slide Time: 07:42)



Contd...

Updating weights connecting output layer and hidden layer

Computation of $\frac{\partial E}{\partial w_{ij}}$:

$$\frac{\partial E(t)}{\partial w_{ij}(t)} = \sum_{i=1}^{n} \frac{\partial E_i(t)}{\partial y_i} \times \frac{\partial y_i}{\partial w_{ij}(t)}$$

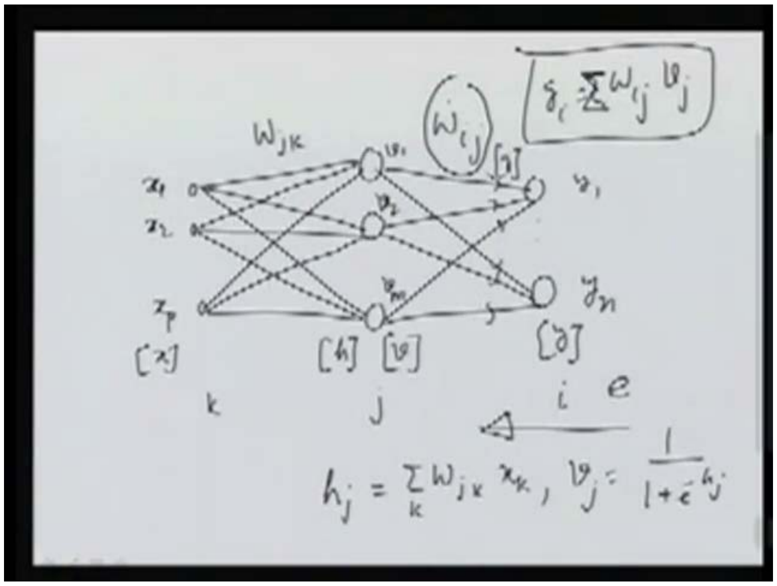
where

$$\frac{\partial E(t)}{\partial w_{ij}(t)} = \frac{1}{2} \times 2(y_i^d - y_i) \times (-1)$$

$$\frac{\partial y_i}{\partial w_{ij}} = \frac{\partial y_i}{\partial g_i} \times \frac{\partial g_i}{\partial w_{ij}}$$

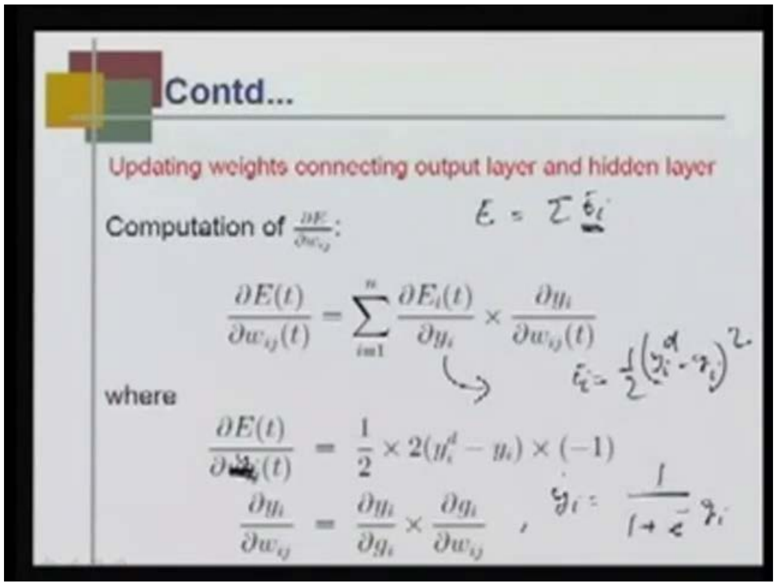
During that, we update the weights connecting the output layer and hidden layer. The weight that we are proposing, which is w_{ij}

(Refer Slide Time: 07:54)



You can see that the weight w_{ij} that we are talking about is this weight. Once I transfer the signal from x to y, then my target is there. I compute the target error here and I back propagate the target error. This is my e here at the target (Refer Slide Time: 08:10 min) and this target error is back propagated. Through back propagation, I update what is w_{ij} – it has to be updated.

(Refer Slide Time: 08:26)



Contd...

Updating weights connecting output layer and hidden layer

Computation of $\frac{\partial E}{\partial w_{ij}}$: $\quad E = \sum E_i$

$$\frac{\partial E(t)}{\partial w_{ij}(t)} = \sum_{i=1}^{n} \frac{\partial E_i(t)}{\partial y_i} \times \frac{\partial y_i}{\partial w_{ij}(t)}$$

$E_i = \frac{1}{2}(y_i^d - y_i)^2$

where

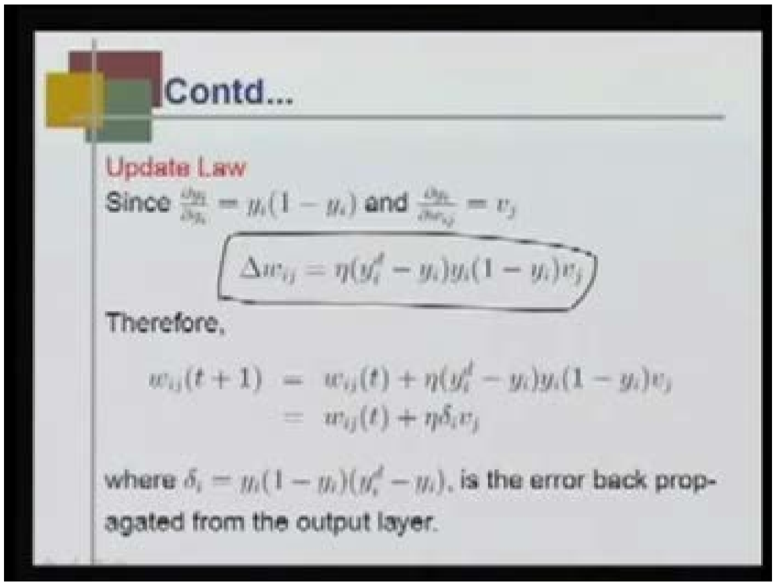
$$\frac{\partial E(t)}{\partial y_i(t)} = \frac{1}{2} \times 2(y_i^d - y_i) \times (-1)$$

$$\frac{\partial y_i}{\partial w_{ij}} = \frac{\partial y_i}{\partial g_i} \times \frac{\partial g_i}{\partial w_{ij}} \quad , \quad y_i = \frac{1}{1+e^{-g_i}}$$

To compute what is the weight update in w_{ij}, we need to compute what is dow E of t upon dow w_{ij} and that gives me the formula dow E_i of t upon dow y_i into dow y_i upon dow w_{ij} of t. How do I compute dow E upon dow w_{ij}? I know that E is summation of E_i. Hence, differentiating E with respect to w_{ij} means I differentiate E_i with respect to w_{ij} and take a sum. Then, individual E_i is a function of y_i. So, I differentiate E_i with respect to dow y_i and then, I differentiate y_i, which is a function of w_{ij}. That is how I compute this partial derivative.

Here, this is dow E upon dow y_i of t, which is half, because, this function E_i is simply y_i d minus y_i whole square – this is E_i. Hence, if you look at this here, it is half into 2 y_i d minus y_i into… when you differentiate with respect to y_i, we get minus 1 here. That is very clear. Now, dow y_i upon dow w_{ij} is dow y_i upon dow g_i because, we found out that y_i is 1 upon 1 plus e to the power of minus g_i. We have already shown that here: y_i is a function of g_i (Refer Slide Time: 10:21). While differentiating, I will obviously differentiate y_i with respect to g_i and then g_i with w_{ij}.

(Refer Slide Time: 10:33)



Contd...

Update Law

Since $\frac{\partial y_i}{\partial g_i} = y_i(1-y_i)$ and $\frac{\partial g_i}{\partial w_{ij}} = v_j$

$$\boxed{\Delta w_{ij} = \eta(y_i^d - y_i)y_i(1-y_i)v_j}$$

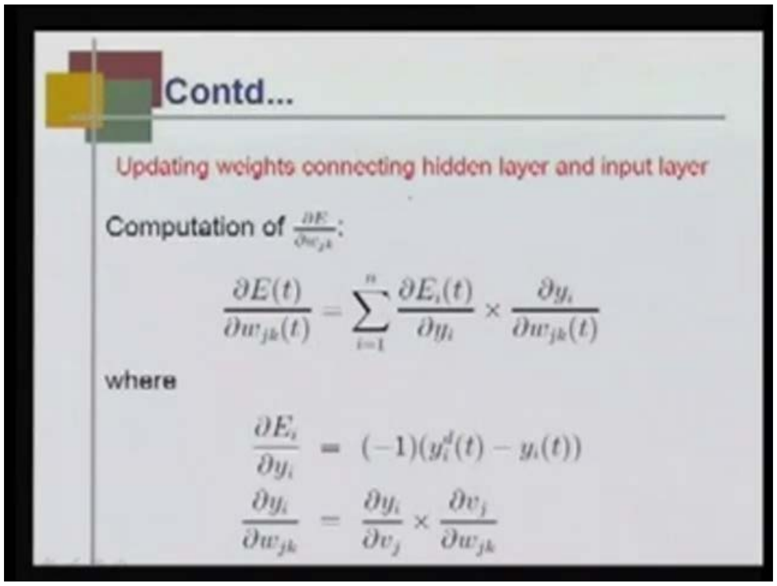
Therefore,

$$\begin{aligned} w_{ij}(t+1) &= w_{ij}(t) + \eta(y_i^d - y_i)y_i(1-y_i)v_j \\ &= w_{ij}(t) + \eta\delta_i v_j \end{aligned}$$

where $\delta_i = y_i(1-y_i)(y_i^d - y_i)$, is the error back propagated from the output layer.

Doing that way, I finally get this expression delta w_{ij} is eta into y_i d minus y_i into y_i into 1 minus y_i into v_j. We wrote the weight update algorithm in a generalized format as w_{ij} of t plus 1 is w_{ij} of t plus eta delta$_i$ v_j and v_j is the input. You can see easily here that when I am updating weights for w_{ij}, v_j is my input to the weights in this layer (Refer Slide Time: 11:10). v_j is the input to the weights in this layer. My error is e, which is… The back propagated error that I am talking of here is represented as delta$_i$ (Refer Slide Time: 11:31). This delta$_i$, which is the error back propagated from the output layer, is defined as y_i into 1 minus y_i into the error at the output, which is y_i d minus y_i. This called the back propagation algorithm. My weights are updated and the new weight is old weight plus eta, error back propagated into the input signal to the connection.

(Refer Slide Time: 12:03)



Contd...

Updating weights connecting hidden layer and input layer

Computation of $\frac{\partial E}{\partial w_{jk}}$:

$$\frac{\partial E(t)}{\partial w_{jk}(t)} = \sum_{i=1}^{n} \frac{\partial E_i(t)}{\partial y_i} \times \frac{\partial y_i}{\partial w_{jk}(t)}$$

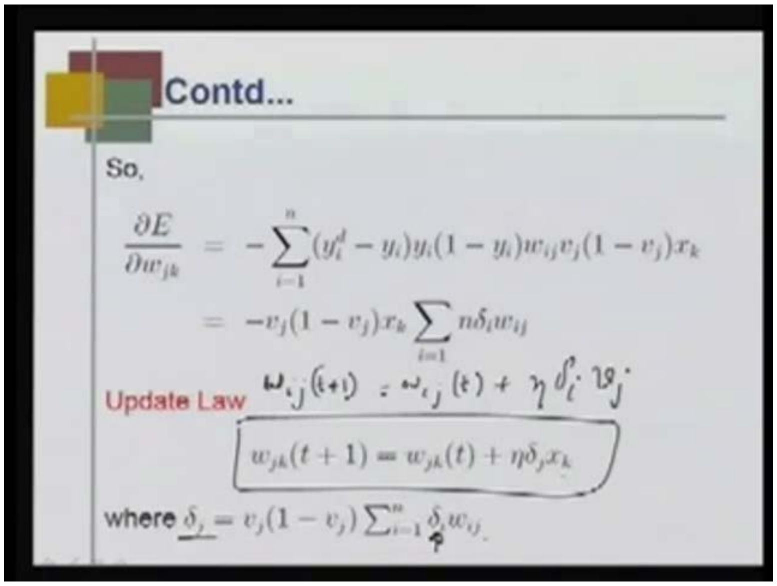
where

$$\frac{\partial E_i}{\partial y_i} = (-1)(y_i^d(t) - y_i(t))$$

$$\frac{\partial y_i}{\partial w_{jk}} = \frac{\partial y_i}{\partial v_j} \times \frac{\partial v_j}{\partial w_{jk}}$$

Similarly, we derived for w_{ij}. Now, we are talking about w_{jk}. w_{jk} was the weight vector or weight matrix between the hidden layer and input layer (Refer Slide Time: 12:20). We have already derived it and I will not go into a detailed discussion. What we did is that when I differentiate E with respect to jk, then I differentiate E_i with respect to y_i and y_i with respect to w_{jk} in a similar manner.

(Refer Slide Time: 12:46)



Contd...

So,

$$\frac{\partial E}{\partial w_{jk}} = -\sum_{i=1}^{n}(y_i^d - y_i)y_i(1-y_i)w_{ij}v_j(1-v_j)x_k$$

$$= -v_j(1-v_j)x_k\sum_{i=1} n\delta_i w_{ij}$$

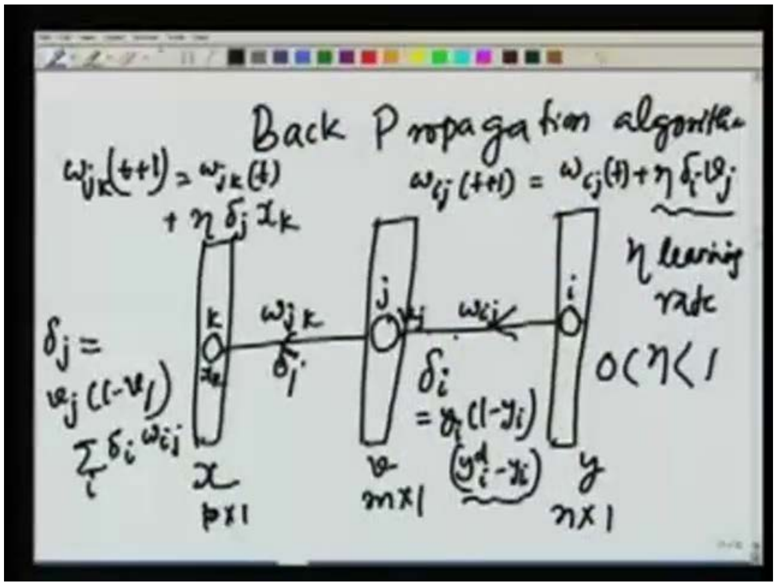
Update Law $\underline{\omega_{ij}(t+1) = \omega_{ij}(t) + \eta\,\delta_i\cdot v_j}$

$$\boxed{w_{jk}(t+1) = w_{jk}(t) + \eta\delta_j x_k}$$

where $\underline{\delta_j} = v_j(1-v_j)\sum_{i=1}^{n}\delta_i w_{ij}$

Ultimately, we finally get a relationship, which is again a generalized form: w_{jk} of t plus 1 is w_{jk} of t plus eta delta_j x_k. This similar format you see, the earlier one was w_{ij} of t plus 1 was w_{ij} of t plus eta delta_j v_j. This is the weight update between output layer and hidden layer and this is the hidden layer and input layer, where the input is x_k in the input layer and delta_j is the error back propagated from the output layer. You see that delta_i is the error back propagated in the output layer and we multiply the corresponding weights w_{ij} and do the summation. The quantity delta_j is the total error back propagated from the output layer to the layer between input layer and hidden layer into v_j into 1 minus v_j. So, delta_j is the back propagated error to the layer that is situated between the hidden layer and input layer. Now, I will summarize what we saw in the back propagation algorithm.

(Refer Slide Time: 14:14)


Back Propagation algorithm
$w_{jk}(t+1) = w_{jk}(t) + \eta \delta_j x_k$
$w_{ij}(t+1) = w_{ij}(t) + \eta \delta_i v_j$
η learning rate
$0 < \eta < 1$
$\delta_j = v_j(1-v_j) \sum_i \delta_i w_{ij}$
$\delta_i = y_i(1-y_i)(y_i^d - y_i)$
w_{jk}
w_{ij}
k
j
i
x
p×1
v
m×1
y
n×1

We have a typical input layer – vector x, which is p by 1 and then a single hidden layer whose output is v, which is m by 1. This is p. Then, there is an output layer whose output is y and this is n by 1. Let us consider a j th computational unit in the hidden layer, i th computational unit in the output layer, and k th computational unit in the input layer. What we saw in the back propagation algorithm is w_{ij} of t plus 1 is w_{ij} of t plus eta delta$_i$ v_j. What is v_j? What is w_{ij}?

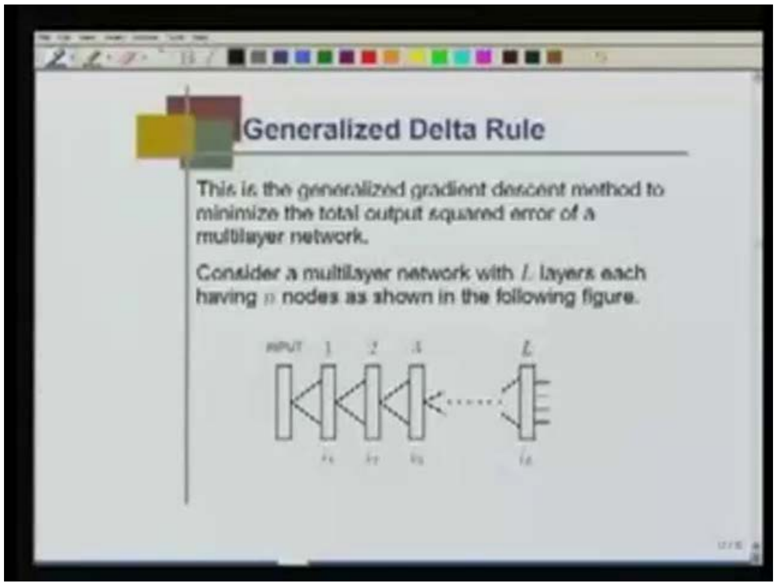
This is my connection – this is w_{ij}. This particular weight is updated based on the error on a single data set, based on the instantaneous update rule. What we do is we update w_{ij} based on its previous value and the connection is…. eta is the learning rate and normally, this value eta is 0 to 1. delta$_i$ is the error back propagated from the output to this output layer. I normally say that this is the second layer and this is the first layer. Let me say that this is the second layer.

We found out delta$_i$, the weight to be y_i into 1 minus y_i into the error that is y_i d minus y_i. This is my error at the output. This multiplied by y_i into 1 minus y_i delta$_i$. v_j is the input – this is v_j (Refer Slide Time: 17:10). If you look at w_{ij}, the input is v_j to this connection and the output of this unit is y_i. Given that, the delta$_i$ has a unique structure. delta$_i$ is y_i into 1 minus y_i into y_i d minus y_i and the update rule also has a unique structure because, v_j is the input to the connection, delta is the error being back propagated through this weight and this is called the delta rule.

The weight update rule has a very simple form, which is delta, which is error back propagated, into v_j. We will see the same thing here. This weight is w_{jk} by convention we have already discussed that and this is my x_k – the output of this unit. The output of this unit is v_j. We saw that the weight update law is w_{jk} of t plus 1 is w_{jk} of t plus eta $delta_j$ into x_k. You see that this and this have the same form and that is why this is called delta rule. Of course, the $delta_j$ has a different value than $delta_k$, but it also has a unique structure. $delta_j$ is v_j into 1 minus v_j into sigma over i $delta_i$ w_{ij}.

I add i, the output (Refer Slide Time: 19:11) 1 to n $delta_i$ w_{ij}, multiply v_j and 1 minus v_j. This is my error being propagated from this unit over this (Refer Slide Time: 19:22). This is the error $delta_j$. So this is $delta_j$. $delta_j$ being back propagated in this particular connection output. It is very important to understand this. $delta_j$ into x_k, where x_k is the input to this connection. You can easily see that the error back propagated in that particular connection weight into the input to that connection weight, if you multiply by the learning rate, that gives you the back propagation rule. It is a very simple rule. Using this delta rule concept, we can write down this back propagation algorithm for any layer in the network. That is what we will write now.

(Refer Slide Time: 20:09)



Generalized Delta Rule

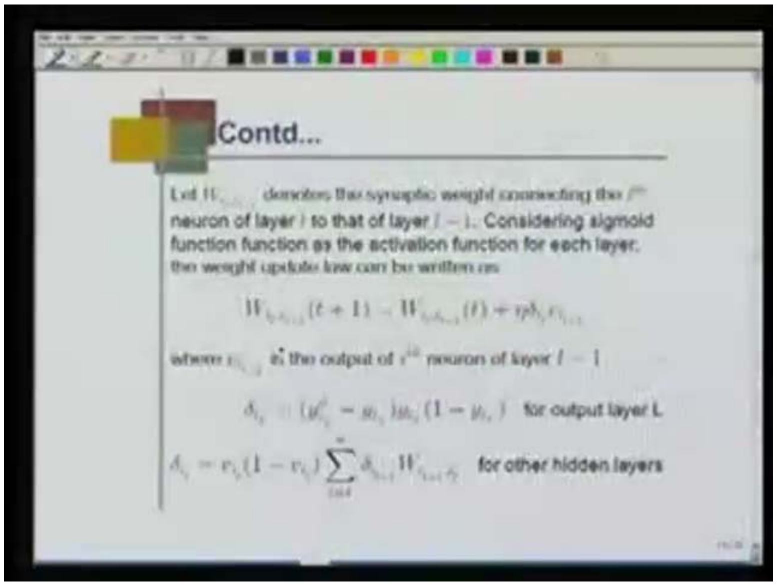
This is the generalized gradient descent method to minimize the total output squared error of a multilayer network.

Consider a multilayer network with L layers each having n nodes as shown in the following figure.

Generalized delta rule: We derived this delta rule for a single hidden layer. Now, we will generalize this delta rule for any number of hidden layers in a feed forward network. That is why it is called a generalized delta rule. You see here we have a multilayered neural network and this multilayered neural network has L layers. This is the input layer, first layer, second layer, third layer and L th layer. These are all feed forward networks, these are all feed forward networks and these are all feed forward networks.

Like we earlier assumed, the indexes for each layer are i, j and k; i is for output layer, j is for hidden layer and k is for input layer. Similarly, here, the index we have generalized i_L for L th layer and similarly, i_i is for i th layer, i_3 is for the third layer, i_2 is for the second layer and obviously, for the input layer, the index is i_0. With this particular convention, we can now easily write down using the same – whatever we discussed here, the normal delta rule. This is called delta rule (Refer Slide Time: 21:37) because, our weight update algorithm consists of the error back propagated the input to the specific connection weight. So, this is the delta rule. Using that, we can now write down the update rule.

(Refer Slide Time: 21:58)



Contd...

Let $W_{i_l i_{l-1}}$ denotes the synaptic weight connecting the i^{th} neuron of layer l to that of layer $l-1$. Considering sigmoid function function as the activation function for each layer, the weight update law can be written as

$$W_{i_l i_{l-1}}(t+1) = W_{i_l i_{l-1}}(t) + \eta \delta_{i_l} v_{i_{l-1}}$$

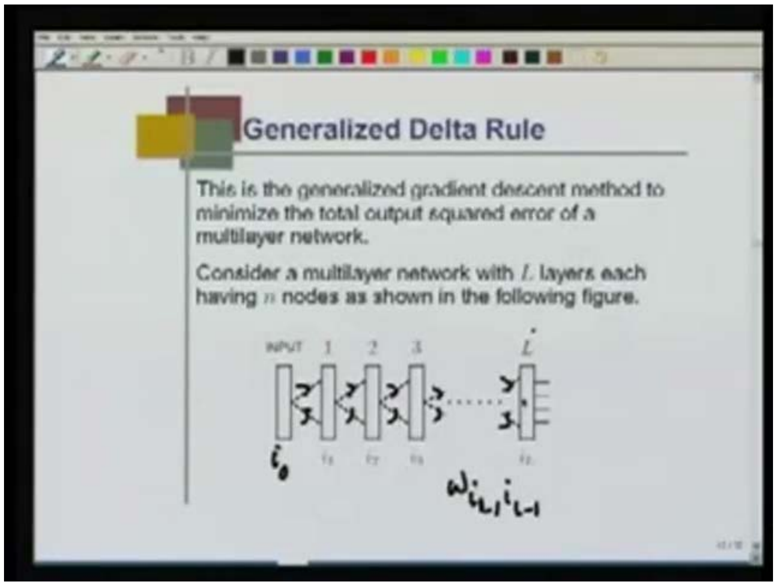
where $v_{i_{l-1}}$ is the output of i^{th} neuron of layer $l-1$

$$\delta_{i_L} = (y^d_{i_L} - y_{i_L}) y_{i_L}(1 - y_{i_L}) \quad \text{for output layer L}$$

$$\delta_{i_l} = v_{i_l}(1 - v_{i_l}) \sum_{i_{l+1}=1}^{n} \delta_{i_{l+1}} W_{i_{l+1} i_l} \quad \text{for other hidden layers}$$

You see here, this is the typical connection weight between l th and l minus 1 th layer.

(Refer Slide Time: 22:15)



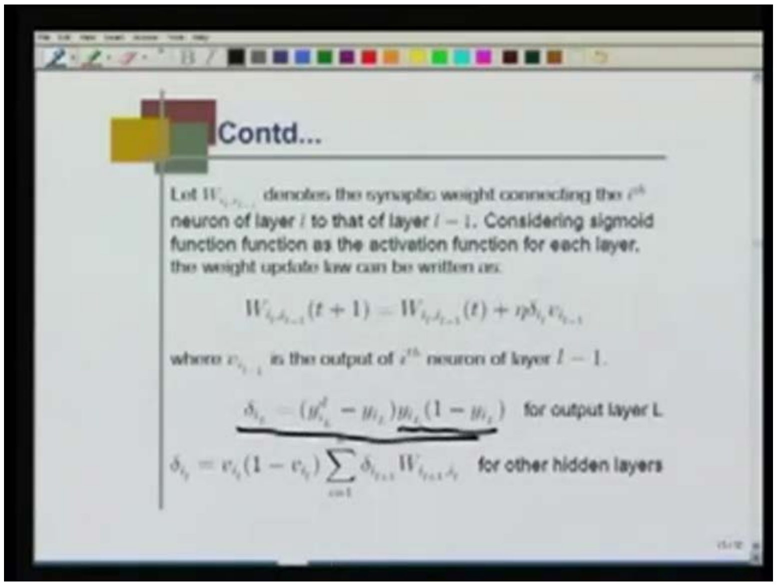
Generalized Delta Rule

This is the generalized gradient descent method to minimize the total output squared error of a multilayer network.

Consider a multilayer network with L layers each having n nodes as shown in the following figure.

What I am talking about here is this is my w$_{il, \text{ il minus 1}}$. These are the weights between the L th layer and the preceding layer. We are talking about how to update the weight in this layer. Once I pass the input (Refer Slide time: 22:42) this network, compute the output at the output layer, compare with the desired data, then I want to update this weight. It is the same formula, that is, eta delta$_{il}$ into v$_{i \text{ l minus 1}}$. This is the input to this connection weight and this is the error being back propagated (Refer Slide Time: 23:08). Obviously, we can easily see that delta$_{il}$ lowercase l and uppercase L are the same – we are not discriminating.

(Refer Slide Time: 23:29)



Contd...

Let $W_{i_l, i_{l-1}}$ denotes the synaptic weight connecting the i^{th} neuron of layer l to that of layer $l-1$. Considering sigmoid function function as the activation function for each layer, the weight update law can be written as:

$$W_{i_l, i_{l-1}}(t+1) = W_{i_l, i_{l-1}}(t) + \eta \delta_{i_l} v_{i_{l-1}}$$

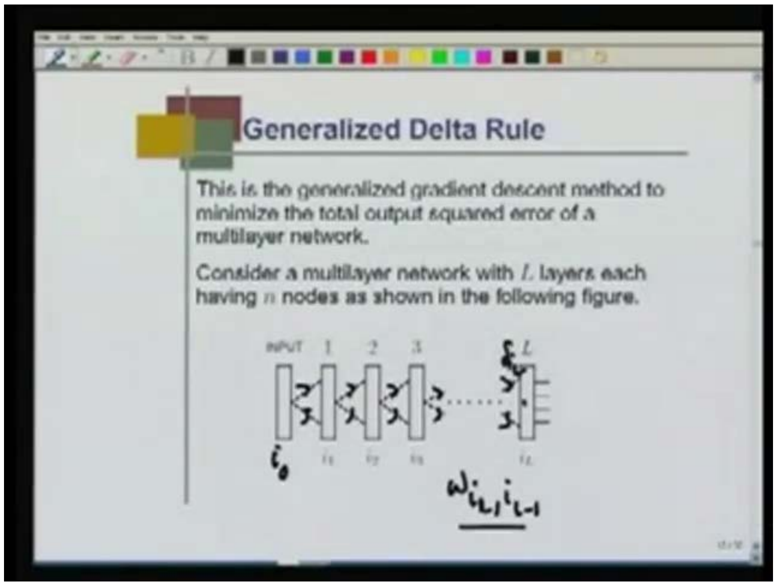
where $v_{i_{l-1}}$ is the output of i^{th} neuron of layer $l-1$.

$$\underline{\delta_{i_L} = (y^d_{i_L} - y_{i_L}) \underline{y_{i_L}(1 - y_{i_L})}}$$ for output layer L

$$\delta_{i_l} = v_{i_l}(1 - v_{i_l}) \sum_{i=1} \delta_{i_{l+1}} W_{i_{l+1} i_l}$$ for other hidden layers

delta$_{il}$ is y $_{il}$ d minus y $_{il}$ into this is our error at the output and you have to multiply this term y$_{il}$ into 1 minus y$_{il}$. That transforms the error at the output to the… when it is back propagated.

(Refer Slide Time: 23:50)



Generalized Delta Rule

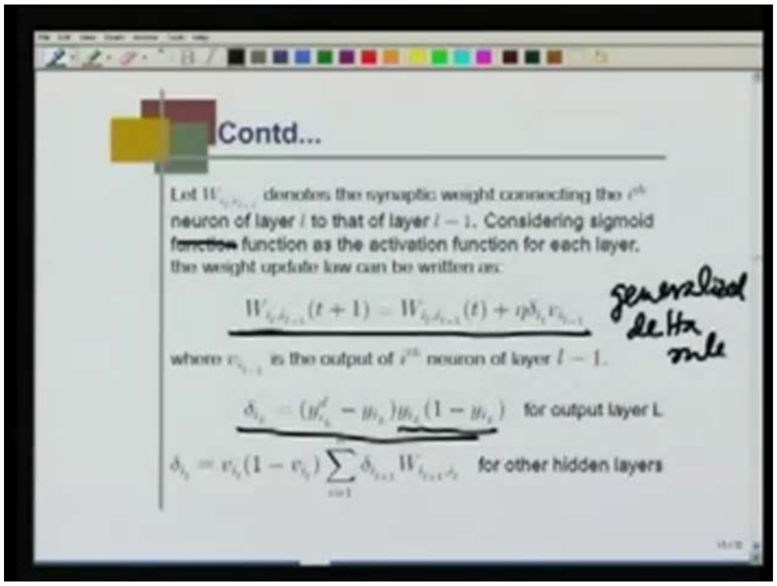
This is the generalized gradient descent method to minimize the total output squared error of a multilayer network.

Consider a multilayer network with L layers each having n nodes as shown in the following figure.

This delta$_{il}$ (Refer Slide Time: 23:48) is here. This is the error back propagated in this immediate layer; that is the L th layer. This is for the output layer. This l represents any layer and delta$_{il}$ is the… for capital L, that is, the output layer. We have made some changes here. Let W$_{il,\ i\ l\ minus\ 1}$

denote the synaptic weight connecting the i th neuron of layer l to that of layer l minus 1. This l is any typical layer. Considering sigmoidal function as the activation function for each layer, the weight update law can be written as….

(Refer Slide Time: 25:05)



Contd...

Let $W_{i_l i_{l-1}}$ denotes the synaptic weight connecting the i^{th} neuron of layer l to that of layer $l-1$. Considering sigmoid ~~function~~ function as the activation function for each layer, the weight update law can be written as:

$$\underline{W_{i_l i_{l-1}}(t+1) = W_{i_l i_{l-1}}(t) + \eta\delta_{i_l} v_{i_{l-1}}}$$

generalized delta rule

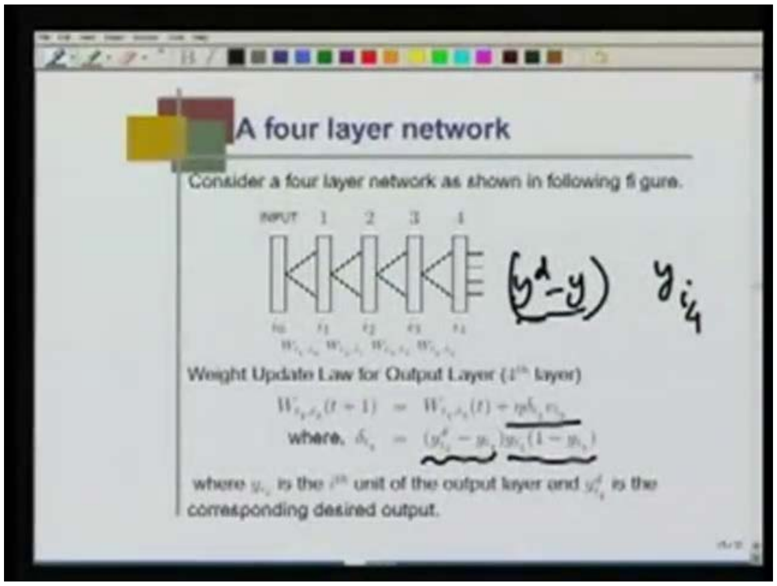
where $v_{i_{l-1}}$ is the output of i^{th} neuron of layer $l-1$.

$$\underline{\delta_{i_L} = (y^d_{i_L} - y_{i_L})\underline{y_{i_L}(1 - y_{i_L})}} \quad \text{for output layer L}$$

$$\delta_{i_l} = v_{i_l}(1 - v_{i_l})\sum_{i=1} \delta_{i_{l+1}} W_{i_{l+1} i_l} \quad \text{for other hidden layers}$$

This is for any layer. This particular weight is the weight between any of these layers (Refer Slide Time: 25:09). For example, if I take L is equal to 2, then this is the layer; if L is equal to 3, then this is the layer; if L is equal to 1, then this is the layer; if l is equal to capital L, then this is the layer. This is the generalized delta rule. Given the delta at the output layer, we can easily compute corresponding delta in all proceeding layers subsequently. Then, we can update all the weights in all the layers using this concept of delta rule. I will give you a simple example. Let us explain this generalized delta rule through another example – we will talk about a four-layer network.

(Refer Slide Time: 26:20)



A four layer network

Consider a four layer network as shown in following figure.

Weight Update Law for Output Layer (4th layer)

$$W_{i_4 i_3}(t+1) = W_{i_4 i_3}(t) + \eta \underline{\delta_{i_4} v_{i_3}}$$

$$\text{where, } \delta_{i_4} = \underbrace{(y^d_{i_4} - y_{i_4})}\underbrace{y_{i_4}(1 - y_{i_4})}$$

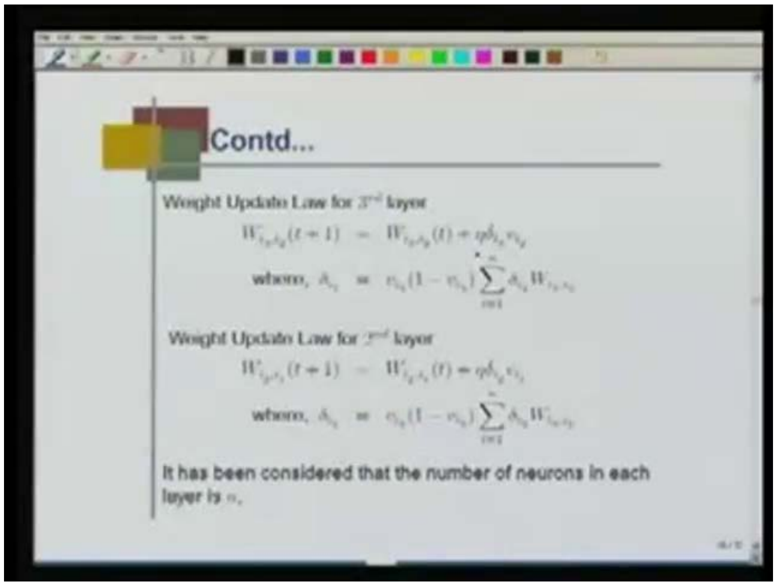
where y_{i_4} is the i^{th} unit of the output layer and $y^d_{i_4}$ is the corresponding desired output.

This is the input layer; this is the first layer, second layer, third layer and fourth layer. Obviously, the weight between the third layer and fourth layer will be represented by $W_{i4,\ i3}$; the weight between the third layer and second layer is $W_{i3,\ i2}$; the weight between the first layer and second layer is $W_{i2,\ i1}$, and the weight between input layer and first layer is $W_{i1,\ i0}$. This is the convention that we are following. Given the error, that is, y d minus y at the output layer, how do we go?

Naturally, the element of y is y_{i4} because, i_4 is the index for the output layer. When I say y is a vector, then y_{i4} is an element of the vector y. So, y is the output vector. This is the weight update law for the output layer, that is, the fourth layer. This is the first layer and then second layer. When I talk in terms of connection weight, this is the first layer of connection weight, second layer of connection weight, third layer of connection weight and fourth layer of connection weight. As many layers we have, we have that many layers of connection weights.

What you are seeing here, as we saw earlier, is that $W_{i4,\ i3}$, the typical weight in the fourth layer, is updated using the generalized delta rule, which is eta into delta, which is the error back propagated, and v_{i3}, which is the input to the connection weight $W_{i4,\ i3}$. What is delta_{i4}? delta_{i4} is obviously y_{i4} d minus y_{i4} into y_{i4} into 1 minus y_{i4}. This is my error (Refer Slide Time: 28:27) that I computed here. This is transformed and we get this because of the sigmoidal activation function. If we have a linear activation function, this particular term will not be there. This is my update rule for the last layer or the output of fourth layer.

(Refer Slide Time: 28:53)



Contd...

Weight Update Law for 3rd layer

$$W_{i_3 i_2}(t+1) = W_{i_3 i_2}(t) + \eta \delta_{i_3} v_{i_2}$$

$$\text{where, } \delta_{i_3} = v_{i_3}(1 - v_{i_3}) \sum_{i=1}^{n} \delta_{i_4} W_{i_4 i_3}$$

Weight Update Law for 2nd layer

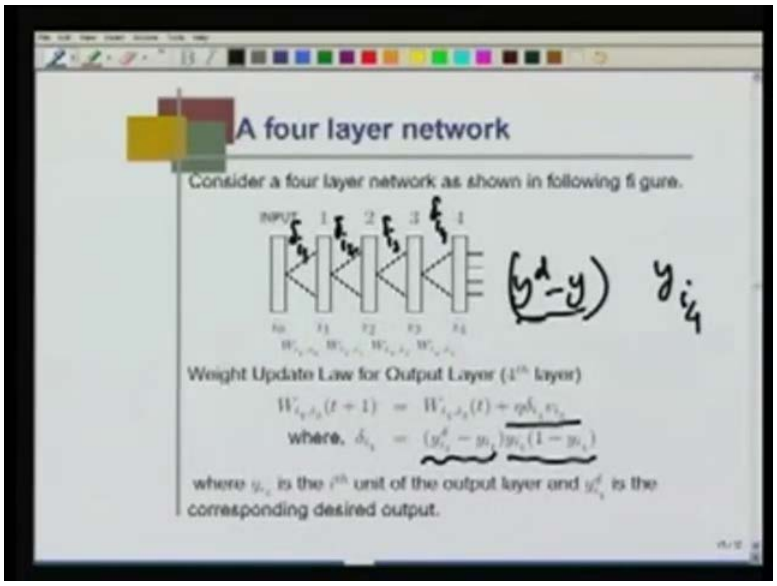
$$W_{i_2 i_1}(t+1) = W_{i_2 i_1}(t) + \eta \delta_{i_2} v_{i_1}$$

$$\text{where, } \delta_{i_2} = v_{i_2}(1 - v_{i_2}) \sum_{i=1}^{n} \delta_{i_3} W_{i_3 i_2}$$

It has been considered that the number of neurons in each layer is n.

Similarly, for the third layer, it is eta delta$_{i3}$ v$_{i2}$.

(Refer Slide Time: 29:00)



A four layer network

Consider a four layer network as shown in following figure.

Weight Update Law for Output Layer (4th layer)

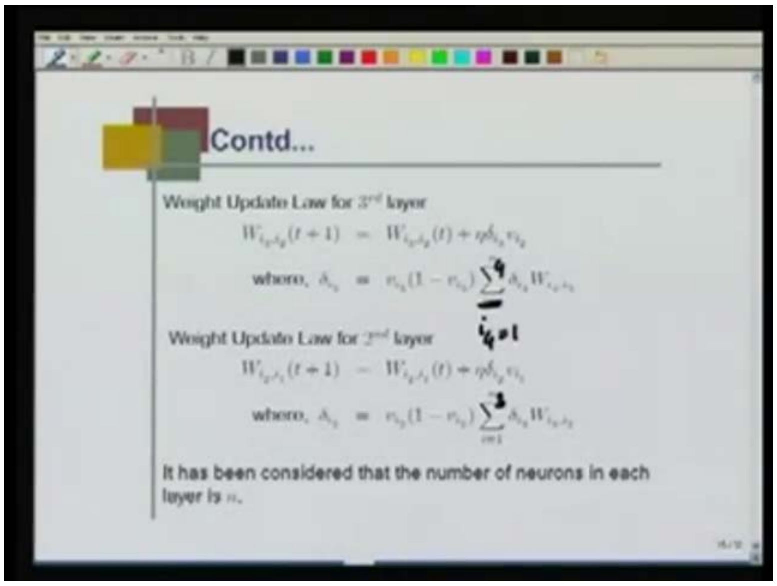
$$W_{i_4 i_3}(t+1) = W_{i_4 i_3}(t) + \overline{\eta \delta_{i_4} v_{i_3}}$$

$$\text{where, } \delta_{i_4} = \underline{(y^d_{i_4} - y_{i_4})}\,\underline{y_{i_4}(1 - y_{i_4})}$$

where y_{i_4} is the i^{th} unit of the output layer and $y^d_{i_4}$ is the corresponding desired output.

This is my delta$_{i4}$ here; this is delta $_{i3}$, delta$_{i2}$ and delta$_{i1}$. I computed what is delta$_{i4}$. Based on delta$_{i4}$, I can compute what delta$_{i3}$ is. This is what I will do.

(Refer Slide Time: 29:19)



Contd...

Weight Update Law for 3rd layer

$$W_{i_3 i_2}(t+1) = W_{i_3 i_2}(t) + \eta \delta_{i_3} v_{i_2}$$

$$\text{where, } \delta_{i_3} = v_{i_3}(1 - v_{i_3}) \sum_{i_4=1}^{n_4} \delta_{i_4} W_{i_4 i_3}$$

Weight Update Law for 2nd layer

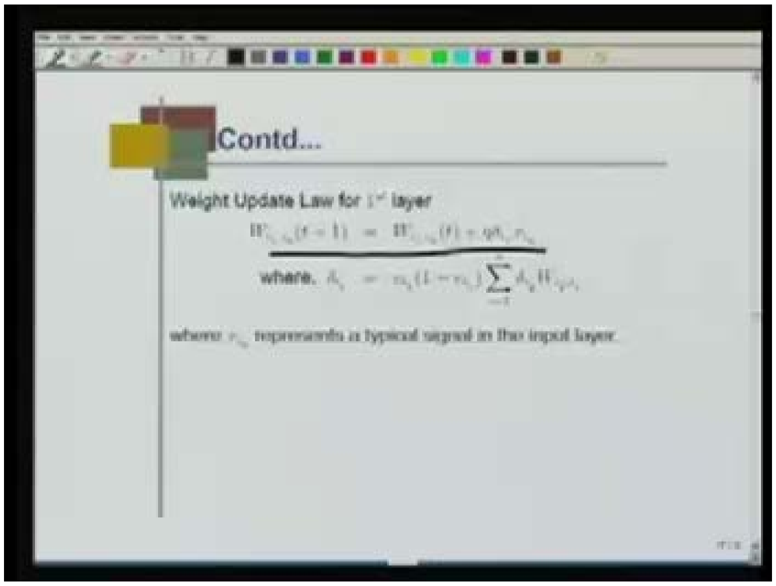
$$W_{i_2 i_1}(t+1) = W_{i_2 i_1}(t) + \eta \delta_{i_2} v_{i_1}$$

$$\text{where, } \delta_{i_2} = v_{i_2}(1 - v_{i_2}) \sum_{i_3=1}^{n_3} \delta_{i_3} W_{i_3 i_2}$$

It has been considered that the number of neurons in each layer is n.

delta$_{i3}$ is v$_{i3}$ into 1 minus v$_{i3}$ into delta i$_4$ W$_{i4,\, i3}$. This is actually i$_4$ equal to 1 to n or the number of units in my L th layer. Let me say that this is n$_4$ and similarly, this is n$_3$. The weight update law for the second layer will be eta delta$_{i2}$ v$_{i1}$. We can easily check it again (Refer Slide Time: 30:08). For this, it is delta$_{i4}$ into vi$_3$; this is delta $_{i3}$ v$_{i2}$; this is delta$_{i2}$ v$_{i1}$ and this is delta$_{i1}$ and x$_{i0}$ because it is the input layer here – x$_{i0}$. This is what we will have now and this is what you can easily see. eta delta$_{i3}$ v$_{i2}$ is for third layer, delta$_{i2}$ v$_{i1}$ is for second layer, and this is your error back propagated (Refer Slide Time: 30:44). delta$_{i2}$ is computed based on delta$_{i3}$ and delta$_{i3}$ is computed based on delta$_{i4}$.

(Refer Slide Time: 30:54)



Contd...

Weight Update Law for 1st layer

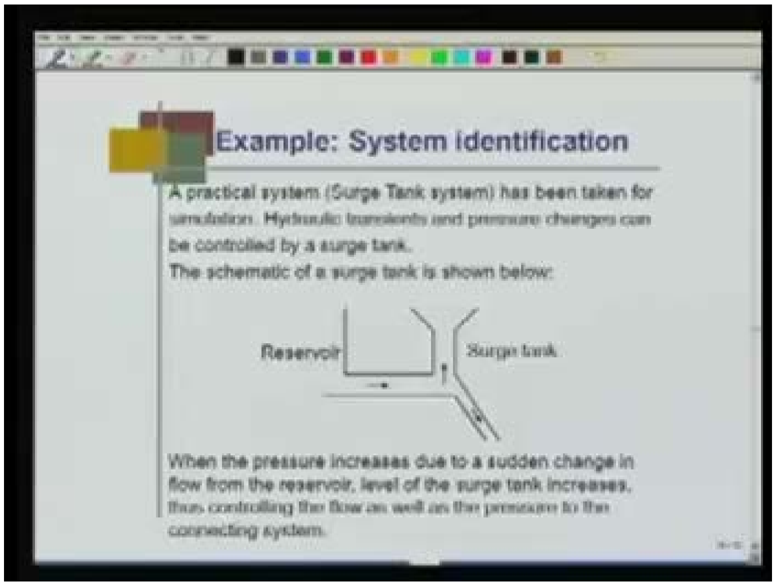
$$W_{i_1 i_0}(t+1) = W_{i_1 i_0}(t) + \eta \delta_{i_1} x_{i_0}$$

where, $\delta_{i_1} = x_{i_1}(1 - x_{i_1}) \sum_{i_2=1}^{n_2} \delta_{i_2} W_{i_2 i_1}$

where x_{i_0} represents a typical signal in the input layer.

Similarly, finally, delta$_{i1}$ is computed based on delta$_{i2}$. This is your first layer. Given any layer, using generalized delta rule, we can write the back propagation algorithm. We do not have to derive again making all those complicated forward response equations and then differentiating the error with respect to each typical weight and finding all those – there is no need actually. This rule is actually generalized and we do not have to compute. We are actually (Refer Slide Time: 31:32) formula. These represent the right formula.

(Refer Slide Time: 31:39)



Example: System identification

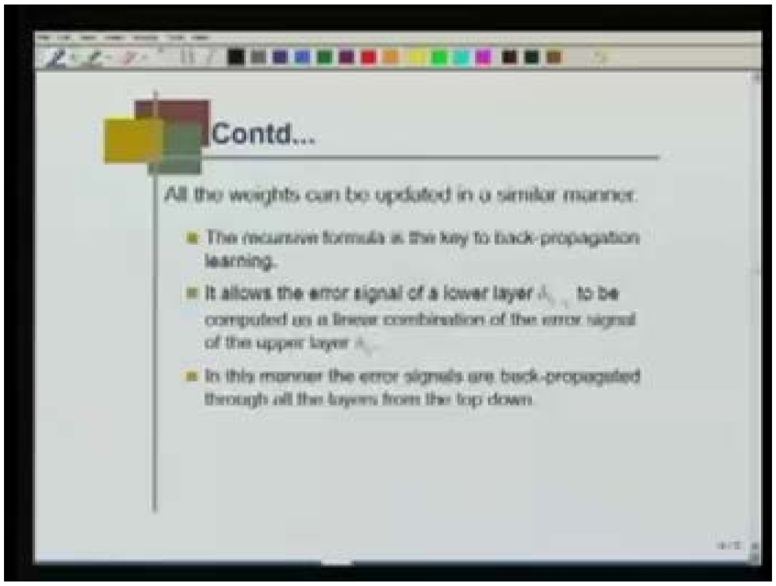
A practical system (Surge Tank system) has been taken for simulation. Hydraulic transients and pressure changes can be controlled by a surge tank.

The schematic of a surge tank is shown below:

When the pressure increases due to a sudden change in flow from the reservoir, level of the surge tank increases, thus controlling the flow as well as the pressure to the connecting system.

We are done with the first part that we wanted to explain today – the generalized delta rule. In the generalized delta rule, the concept was that we wanted a recursive formula for a back propagation learning algorithm.

(Refer Slide Time: 32:04)



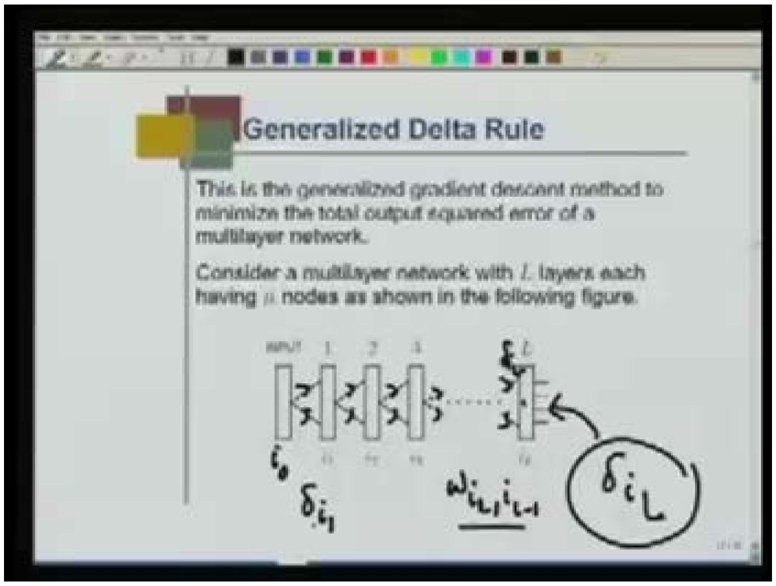
Contd...

All the weights can be updated in a similar manner.

- The recursive formula is the key to back-propagation learning.
- It allows the error signal of a lower layer δ_{q-1} to be computed as a linear combination of the error signal of the upper layer δ_q
- In this manner the error signals are back-propagated through all the layers from the top down.

It allows the error signal of a lower layer to be computed as a linear combination of the error signal of the upper layer. In this manner, the error signals are back propagated through all the layers from the top down, that is, from the last layer -the output layer, backward until we reach the input layer. We can compute the error back propagation, which is in terms of delta, which is here.

(Refer Slide Time: 32:38)



Generalized Delta Rule

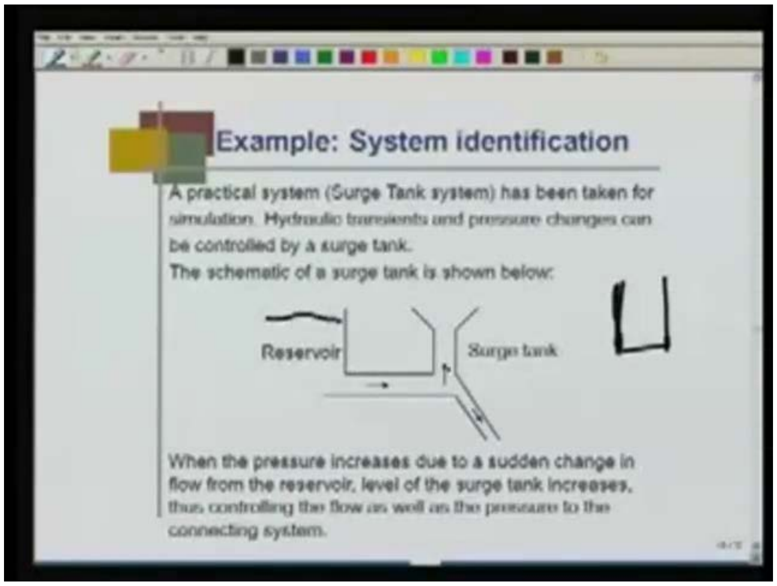
This is the generalized gradient descent method to minimize the total output squared error of a multilayer network.

Consider a multilayer network with L layers each having n nodes as shown in the following figure.

We have to compute here what is delta $_{iL}$ for the output layer and based on that, we finally compute what is delta i_1. delta$_{i1}$ is computed based on delta$_{i2}$, delta$_{i2}$ is computed based on delta$_{i3}$, delta$_{i3}$ is computed based on delta$_{i4}$ and so on. Finally, delta $_{iL \text{ minus } 1}$ is computed based on what is delta $_{iL}$. The general form is that this is the original form (Refer Slide Time: 33:10). This is a typical weight between the l th layer and l minus 1 th layer.

The update rule or delta w is eta delta$_{il}$ into v$_{i\ l \text{ minus } 1}$. What you are seeing is eta delta$_{il}$ v$_{il \text{ minus } 1}$. This is the input to the connection and this is the error back propagated to the layer l. Using this concept, we will now demonstrate the application of the back propagation algorithm for system identification. Since it is a control course, you would always use system identification while demonstrating any application of a specific neural network. A practical system, the surge tank system has been taken for simulation. We will identify the surge tank system using a neural network or we will model the surge tank. So, what is the surge tank?

(Refer Slide Time: 34:27)



Example: System identification

A practical system (Surge Tank system) has been taken for simulation. Hydraulic transients and pressure changes can be controlled by a surge tank.
The schematic of a surge tank is shown below:

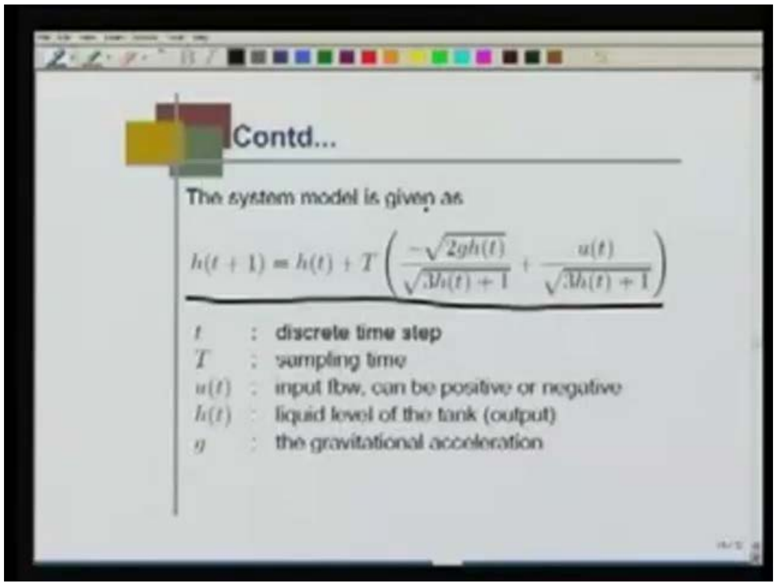
When the pressure increases due to a sudden change in flow from the reservoir, level of the surge tank increases, thus controlling the flow as well as the pressure to the connecting system.

A surge tank- what you are looking at is that. Normally, a surge tank is used to control the hydraulic transients and pressure changes. You see this is a big reservoir. In this reservoir, the water level sometimes increases suddenly or suddenly decreases, but this reservoir is connected particularly to a hydro power plant, where the turbines have to move in a specific speed. So, the flow rate here has to be constant. If the water level suddenly increases or suddenly decreases in this reservoir, then this flow rate will be affected. To maintain the flow rate as constant, the surge tank is used.

What happens? Whatever the disturbance in the reservoir level, it can be controlled using this surge tank. The extra pressure you have, the extra flow can be pushed to the surge tank. Normally, the surge tank has a nonlinear structure level. For a surge tank, the structure is like this. This is a cylindrical structure. That means everywhere, the diameter of this surge tank… for example, if this is circular, then the diameter is constant. But if the surge tank has a nonlinear structure, then given the specific flow into this surge tank, the water level in the surge tank to the flow rate into the surge tank will bear a nonlinear relationship.

We will now try to model this surge tank, because, unless we know how to model the surge tank, we cannot design; because we must know how the water level in this surge tank is increasing or decreasing, given a specific water flow into this surge tank. Based on that, we will design a surge tank and a specific dimension of the surge tank, that is, how the diameter will vary – linearly or nonlinearly, etc. We are just taking a nonlinear surge tank model. This is a schematic diagram of a surge tank. When the pressure increases due to sudden change in the flow from the reservoir, the level of the surge tank increases; thus, controlling the flow as well as pressure to the connecting system; this flow (Refer Slide Time: 37:27) is controlled accordingly.

(Refer Slide Time: 37:32)



Contd...

The system model is given as

$$h(t+1) = h(t) + T\left(\frac{-\sqrt{2gh(t)}}{\sqrt{3h(t)+1}} + \frac{u(t)}{\sqrt{3h(t)+1}}\right)$$

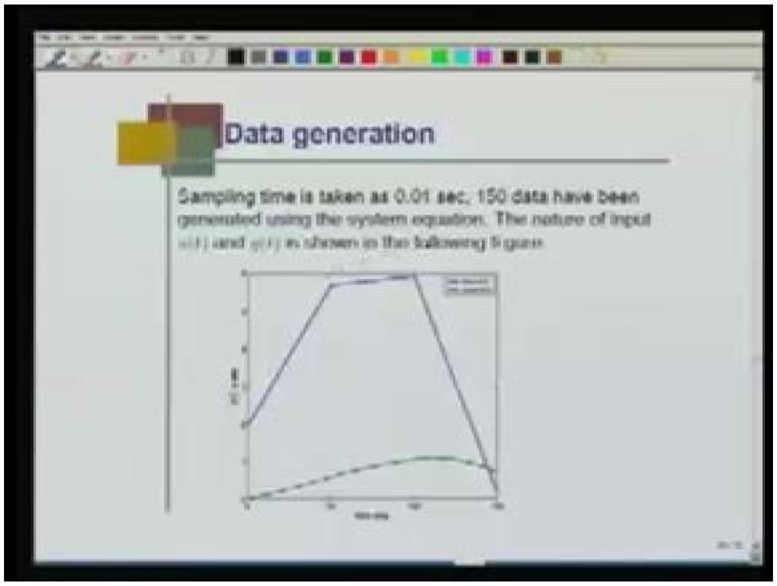
t	:	discrete time step
T	:	sampling time
$u(t)$	:	input flow, can be positive or negative
$h(t)$	:	liquid level of the tank (output)
g	:	the gravitational acceleration

Typically, we are not interested now in how we derive the model of a surge tank. What we are interested in is given a nonlinear structure-based surge tank, this is the dynamics of the surge tank, that is, h of t plus 1 is equal to h of t plus T, which is the sampling time, and this particular nonlinear … you can see easily that this is a nonlinear function (Refer Slide Time: 38:05), because, this is square root of h and also, u is multiplied with 1 upon square root of h and hence, this is a nonlinear function.

t is the discrete time step, that is, the sample instant per second (Refer Slide Time: 38:25), T is the sampling time, u of t is input flow, h of t is the liquid level in the surge tank and g is the acceleration due to gravity. This is our model of the surge tank. Why I am considering this model is because, in simulation, we do not collect data from the actual surge tank – we collect data from a mathematical model in simulation. Using that data, we will represent those data in terms of neural network model. This neural network is the back propagation network that we have just earned.

What we have done is that we have taken this model and we generated data using this model, not using an actual surge tank – an actual surge tank is there in a field, probably in a hydro power plant. We want to demonstrate the application of a back propagation algorithm to a specific system modeling. This is the mathematical model of the surge tank and using this mathematical model, we generate data. Using that data, we create a back propagation network. What we have done here is that we have taken sampling time capital T is 0.01 in the mathematical model.

(Refer Slide Time: 40:02)



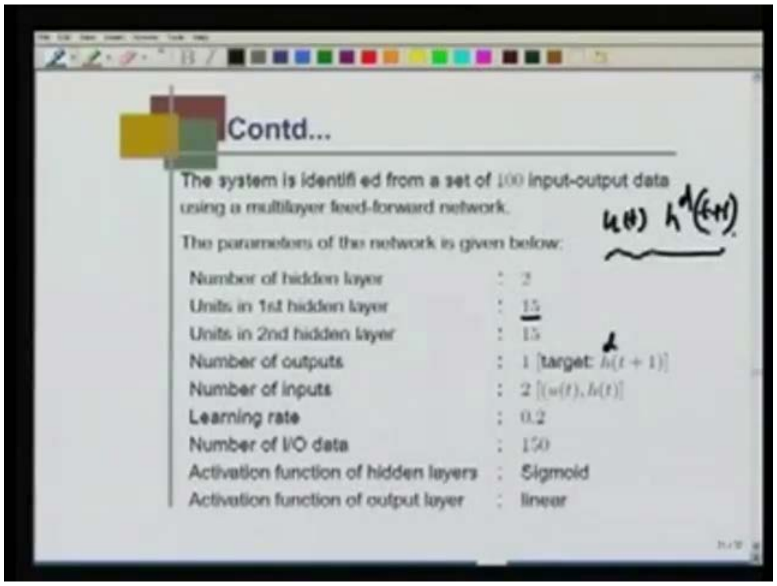
Data generation

Sampling time is taken as 0.01 sec, 150 data have been generated using the system equation. The nature of input u(t) and y(t) is shown in the following figure

150 data have been generated using the system equation. We saw that equation. This is the equation (Refer Slide Time: 40:06). Using this equation, we generated 150 data. What is u of t? Given u of t, what is h of t plus 1? This is our input that we have selected and this is the corresponding output. In that equation, you set T equal to 0.01 and g is the normal acceleration due to gravity, which is 9.81 meter per Second Square. Once you give those parameters, here all the others are known.

T is given as 0.01, you are giving g, so it is simply u of t and h of t. You have given u of t is equal to (Refer Slide Time: 40:52) using this particular equation. We used the Runge–Kutta fourth order to generate the response. This response is generated using Runge–Kutta fourth order equation. Actually, there is no need, this is a discrete time recursive equation. If it is a differential equation, then we would have used the Runge–Kutta equation but it is a simple recursive equation and there is no need for the Runge-Kutta equation. We generated this data given this input and then we selected a network structure.

(Refer Slide Time: 41:36)



Contd...

The system is identifi ed from a set of 100 input-output data using a multilayer feed-forward network.

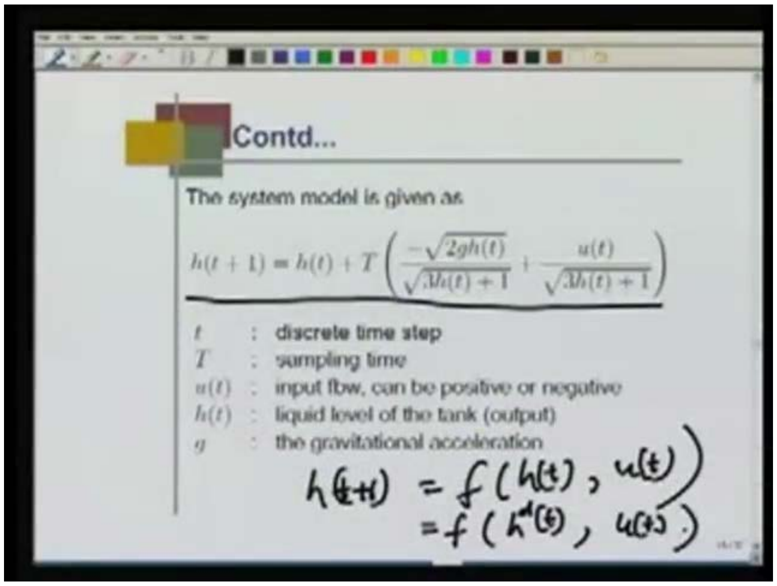
$u(t)$ $h^d(t+1)$

The parameters of the network is given below:

Number of hidden layer	: 2
Units in 1st hidden layer	: $\underline{15}$
Units in 2nd hidden layer	: 15
Number of outputs	: 1 [target: $h^d(t+1)$]
Number of inputs	: 2 [$(u(t), h(t)$]
Learning rate	: 0.2
Number of I/O data	: 150
Activation function of hidden layers	: Sigmoid
Activation function of output layer	: linear

We took a back propagation network having two hidden layers. Each hidden layer has 15 neurons or computational units and the number of outputs is the… this is desired (Refer Slide Time: 41:57). We represent data in this term, u of t and h desire t plus 1. We get this from the system equation. This is my desired output: h d t plus 1, given u of t. But you can see here, how we select.

(Refer Slide Time: 42:30)



Contd...

The system model is given as

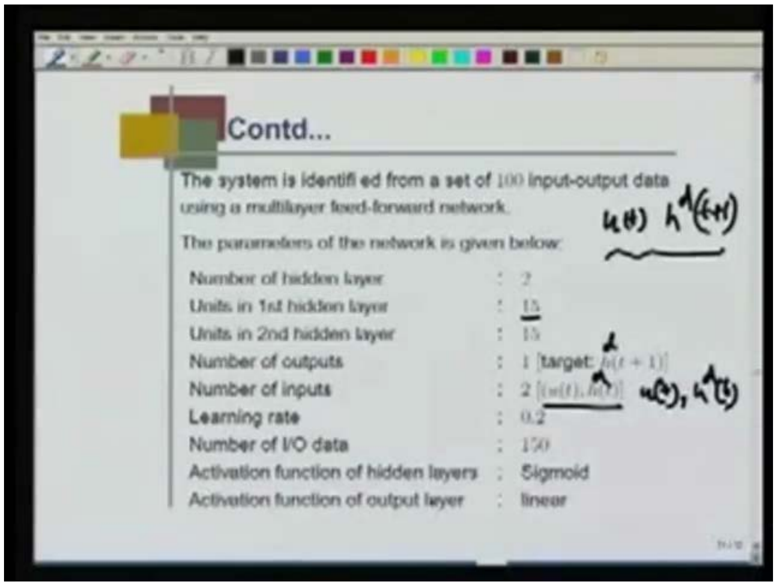
$$h(t+1) = h(t) + T\left(\frac{-\sqrt{2gh(t)}}{\sqrt{3h(t)+1}} + \frac{u(t)}{\sqrt{3h(t)+1}}\right)$$

t	:	discrete time step
T	:	sampling time
$u(t)$	:	input flow, can be positive or negative
$h(t)$	:	liquid level of the tank (output)
g	:	the gravitational acceleration

$$h(t+1) = f(h(t), u(t))$$
$$= f(h^d(t), u(t))$$

You see here in the back propagation network feed forward network, if I give u of t, I cannot complete h of t plus 1. You can easily see that h of t plus 1 is a function of h of t as well as u of t; h of t plus 1 is a function of h of t and u of t. Obviously, I take this h of t from the system, because at instant t, I am able to measure what is the liquid level in the surge tank; I am measuring that and so I know the actual value of h d t; u of t is the flow rate that is going to the surge tank, which I am able to measure. Given the actual liquid level of the surge tank at time t and the flow rate u of t, I have to predict what h of t plus 1 is. This is my model.

(Refer Slide Time: 43:49)



Contd...

The system is identifi ed from a set of 100 input-output data using a multilayer feed-forward network.

The parameters of the network is given below:

Number of hidden layer	: 2
Units in 1st hidden layer	: $\underline{15}$
Units in 2nd hidden layer	: 15
Number of outputs	: 1 [target: $h^d(t+1)$]
Number of inputs	: 2 $\underline{[(u(t), \hat{h}(t)]}$
Learning rate	: 0.2
Number of I/O data	: 150
Activation function of hidden layers	: Sigmoid
Activation function of output layer	: linear

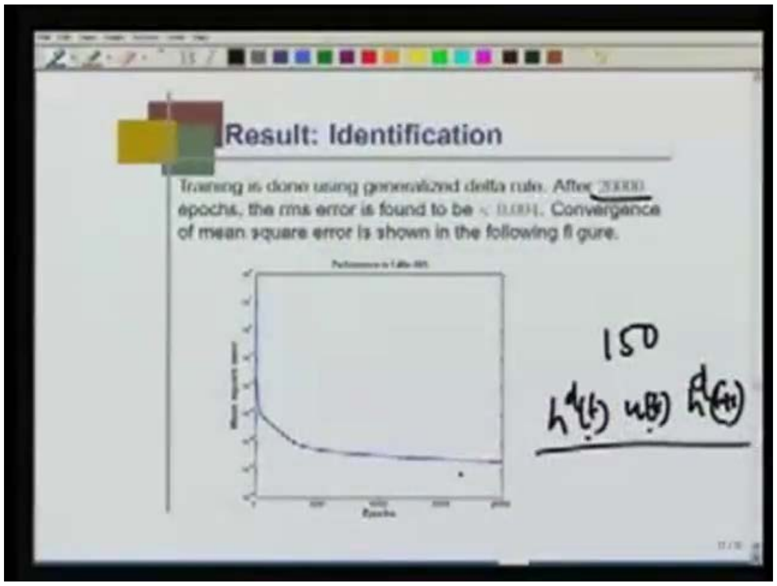
$\underset{\sim\!\!\sim}{u(t)\ h^d(t+1)}$

$u(t), h^d(t)$

Obviously, for my system, the input is u of t and h d t, which is written here, this is actually d here (Refer Slide Time: 43:59) the desired, which has been computed from the actual system. The number of inputs that I have is 2 and output is my h desired t plus 1. Given u of t and h d t, I must predict what is h d t plus 1 – that is the objective. We have fixed the learning rate to be 0.2, number of input/output data is 150, the activation function for hidden layers is sigmoidal and we have taken the activation function of the output layer to be linear. You have two hidden layers, so (Refer Slide time: 44:34) sigmoid and the output (Refer Slide Time: 44:36), but you can select your own architecture and do the same thing.

In fact, this can be easily done using a single hidden layer and output layer with sigmoidal activation function – we can easily do it. What we do is that we have 150 training data, we selected network architecture; we observed what the input to the network is and what the output is and then we update the equation recursively – update the weights. In the beginning, all the weights in the network were randomly initialized – we took very small values.

(Refer Slide Time: 45:26)

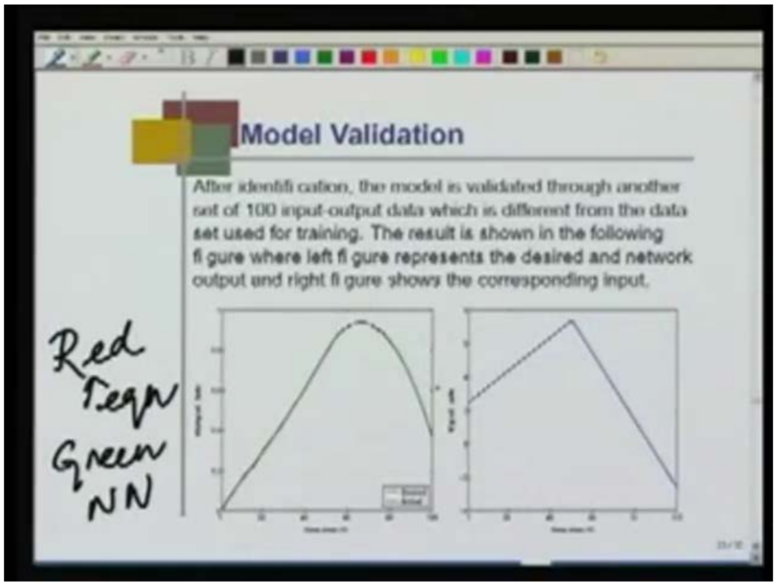


Result: Identification

Training is done using generalized delta rule. After 20000 epochs, the rms error is found to be < 0.004. Convergence of mean square error is shown in the following figure.

You will see that the error has been reduced to less than 0.004 after 20,000 epochs. What is an epoch? Each epoch is 150; that is, in my data set, I have 150 different sets of data. One data is h d t u t and h d t plus 1. This is my single data set at t and t is equal to 1 to 150. Like that, I have 150 data sets. I give to my network u t and h d t, compute what is h d t plus 1, compare it with h d t plus 1, back propagate the error and update the weight; doing that, I am able to reduce or make my cost function 0.004.

(Refer Slide Time: 46:26)



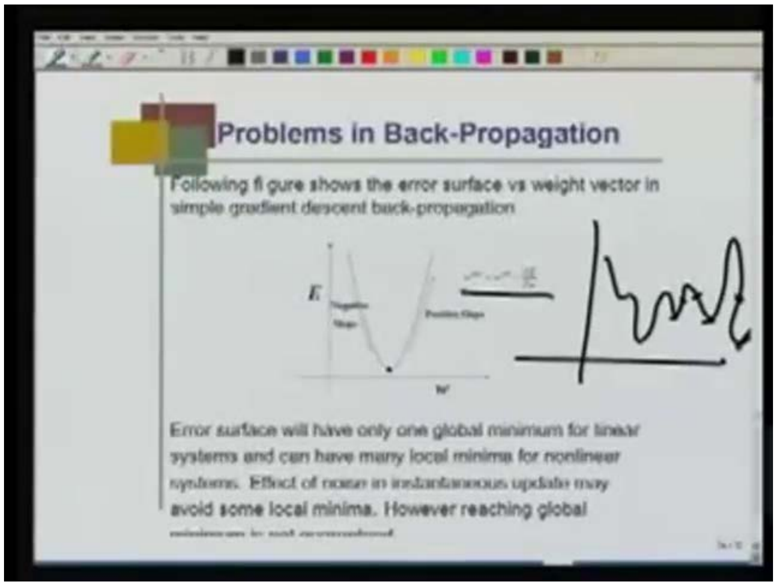
Model Validation

After identification, the model is validated through another set of 100 input-output data which is different from the data set used for training. The result is shown in the following figure where left figure represents the desired and network output and right figure shows the corresponding input.

After training is over, what did we do? We give to the network a different control input and compare it with the actual output, that is, this control input (Refer Slide Time: 46:41 to 46: 45 min) was given to this equation and then we computed what is the output. This is actual output based on this equation. We then gave it to the neural network that has been trained. Once we gave this input to the neural network, the response and the actual output based on the equation are actually matching – you can see that.

You can see that there is a red line here and there is a green line. The red line is the desired based on the equation. Red is computed from the equation and green is from the neural network. Both the red and green are very much following each other. In essence, we can say that the actual system identification has been done; that is, the equation that represents a model of a specific surge tank has been again represented using a back propagation network having two hidden layers. Now, we will look into some other aspects of back propagation.

(Refer Slide Time: 48:08)



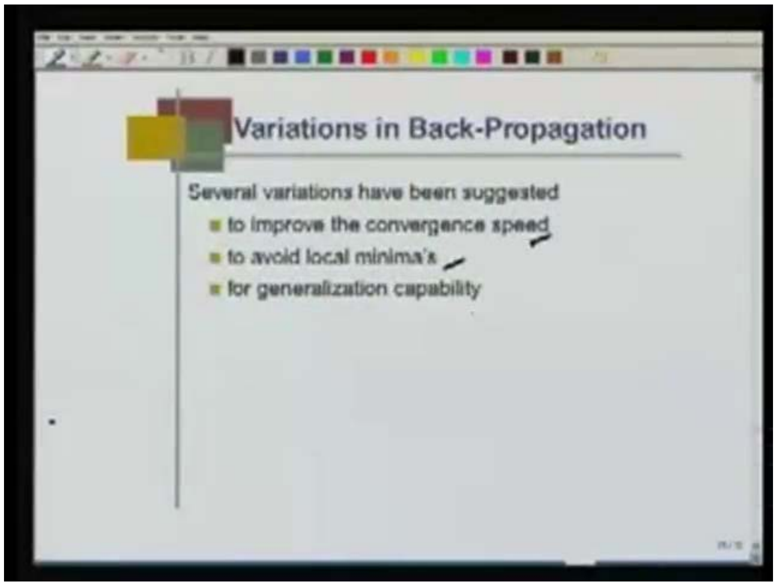
Problems in Back-Propagation

Following figure shows the error surface vs weight vector in simple gradient descent back-propagation

Error surface will have only one global minimum for linear systems and can have many local minima for nonlinear systems. Effect of noise in instantaneous update may avoid some local minima. However reaching global minimum is not guaranteed

As I already told you, some of the problems in back propagation are that the normal back propagation can only optimize if my cost function is a single minima. Normally, in a cost function, you will have many local minima as well as global minima and so, I may reach here or I may reach here. If I start from here (Refer Slide Time: 48:31), I will reach here; if my initial weights are here, then I will reach here, because, that is the gradient descent; if my initial weights are here, I will reach here. This is the limitation of back propagation – reaching global minimum in back propagation is not guaranteed.

(Refer Slide Time: 48:56)



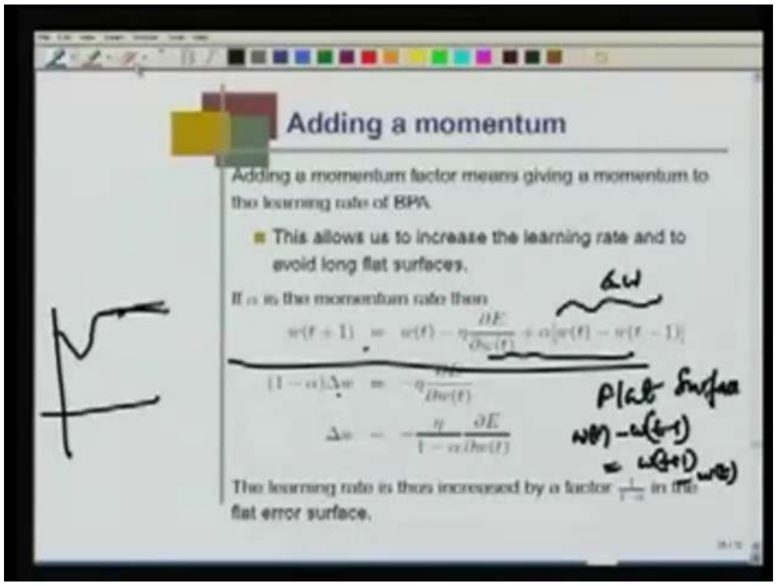
Variations in Back-Propagation

Several variations have been suggested

- to improve the convergence speed
- to avoid local minima's
- for generalization capability

That is why researchers have introduced several variations in the back propagation algorithm. First is to improve the convergence speed, then avoid local minima and the generalizing capability. Generalizing capability means that if I have trained my network for a specific data set, it should be able to predict what the output is for a new data set; that is called generalization. One of the ways to improve the convergence speed is by adding a momentum.

(Refer Slide Time: 49:29)



Adding a momentum

Adding a momentum factor means giving a momentum to the learning rate of BPA

- This allows us to increase the learning rate and to avoid long flat surfaces.

If α is the momentum rate then

$$w(t+1) = w(t) - \eta\frac{\partial E}{\partial w(t)} + \alpha[w(t) - w(t-1)]$$

$$(1-\alpha)\Delta w = -\eta\frac{\partial E}{\partial w(t)}$$

$$\Delta w = -\frac{\eta}{1-\alpha}\frac{\partial E}{\partial w(t)}$$

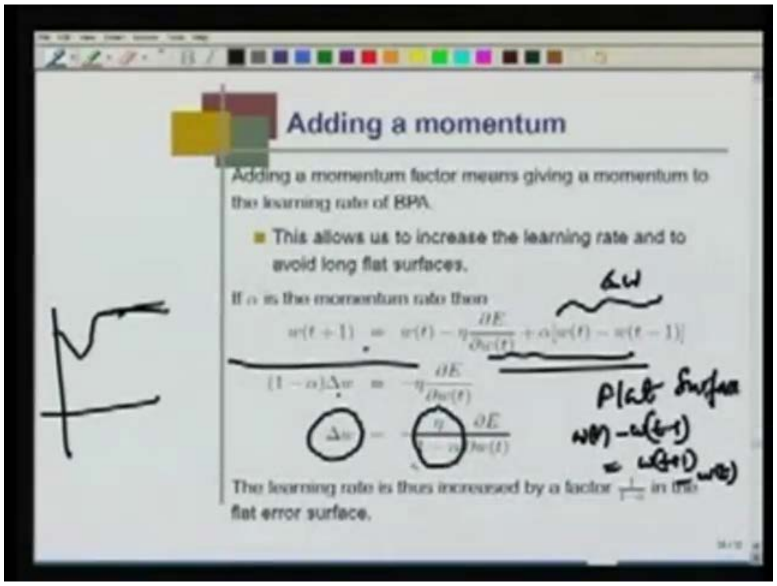
The learning rate is thus increased by a factor $\frac{1}{1-\alpha}$ in the flat error surface.

How do we add a momentum? This is our equation. It is actually a heuristic approach, but we can analyze this equation just immediately. This is our normal equation: w t plus 1 is w of t minus eta into del E by del w of t – this is our normal gradient descent. Then, I have added a momentum term (Refer Slide Time: 49:48), which is alpha into w of t minus w of t minus 1. When I add this, this is heuristic and there is no derivation here – we have not derived this particular term; this is heuristic.

Let us see what happens. Objectively analyze this equation in a flat surface. What is a flat surface? In a flat surface, w of t minus w of t minus 1 is the same as w of t plus 1 minus w of t; this is called a flat surface. Say for example, my error curve is sometimes like this. I am moving like this (Refer Slide Time: 50:33). This is a flat error surface. If I am here, I do not want to stay in the flat surface, I want to come back to the value, I must go towards the value (Refer Slide Time: 50:48) and so, my speed should be very fast in this zone.

In that case, w of t plus 1 minus w of t is the same as w of t minus w of t minus 1, which is delta$_w$. If that is the case, I can rewrite this equation. If I rewrite this equation, this is 1 minus alpha into delta$_w$ and delta$_w$ is w of t plus 1 minus w of t and this is also the same – delta$_w$. This is my delta$_w$ (Refer Slide Time: 51:23) and w of t plus 1 minus w of t is also delta$_w$. So, 1 minus alpha delta$_w$ is minus eta into del E upon del w t.

(Refer Slide Time: 51:37)



Adding a momentum

Adding a momentum factor means giving a momentum to the learning rate of BPA.

- This allows us to increase the learning rate and to avoid long flat surfaces.

If α is the momentum rate then

$$w(t+1) = w(t) - \eta\frac{\partial E}{\partial w(t)} + \alpha\underbrace{[w(t) - w(t-1)]}_{}$$

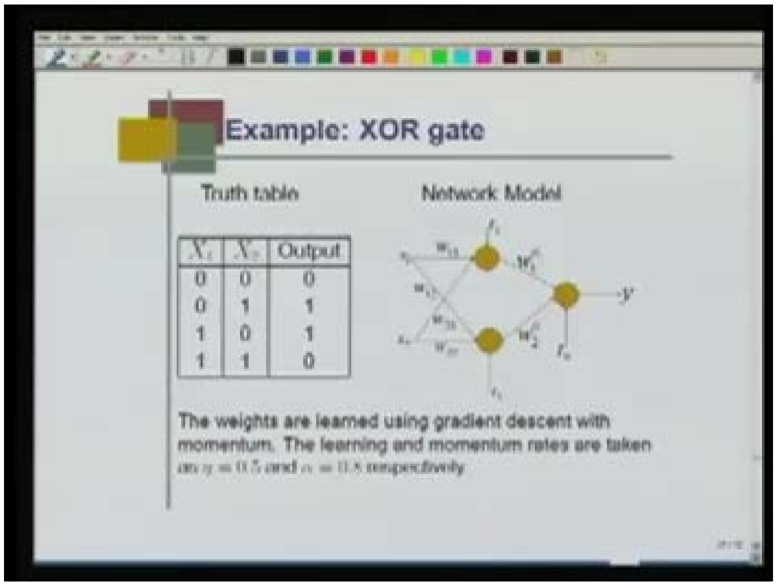
$$(1-\alpha)\Delta w = -\eta\frac{\partial E}{\partial w(t)}$$

$$\Delta w = -\frac{\eta}{1-\alpha}\frac{\partial E}{\partial w(t)}$$

The learning rate is thus increased by a factor $\frac{1}{1-\alpha}$ in the flat error surface.

Look at this equation. I can write down what is my weight update delta$_w$, the increment in weight. It is minus eta upon 1 minus alpha into delta E by del$_w$ t. Normally, delta$_w$ is eta del E upon del$_w$ t, but utilizing this heuristic term, I am able to increase the learning rate by a term called eta by 1 minus alpha and alpha is less than 1. Obviously, if I say alpha is 0.5, then 1 minus 0.5 is 0.5 and the effective increase in the learning rate is twice. If I make alpha 0.9, then 1 minus 0.9 is 0.1 and so, the effective eta is now 10 times the actual eta. The learning rate is thus increased by a factor 1 by 1 minus alpha in the flat error surface. This is how we can increase the convergence speed and this is an example.

(Refer Slide Time: 52:46)



Example: XOR gate

Truth table

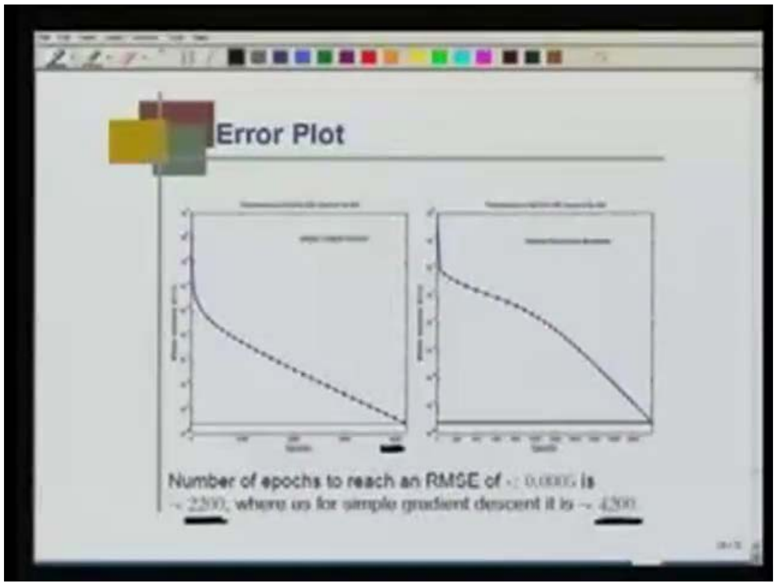
X_1	X_2	Output
0	0	0
0	1	1
1	0	1
1	1	0

Network Model

The weights are learned using gradient descent with momentum. The learning and momentum rates are taken as $\eta = 0.5$ and $\alpha = 0.8$ respectively

This is an XOR network. In the XOR network, we have taken eta equal to 0.5 ;this is the learning rate, and alpha, the momentum rate, is 0.8.

(Refer Slide Time: 53:01)



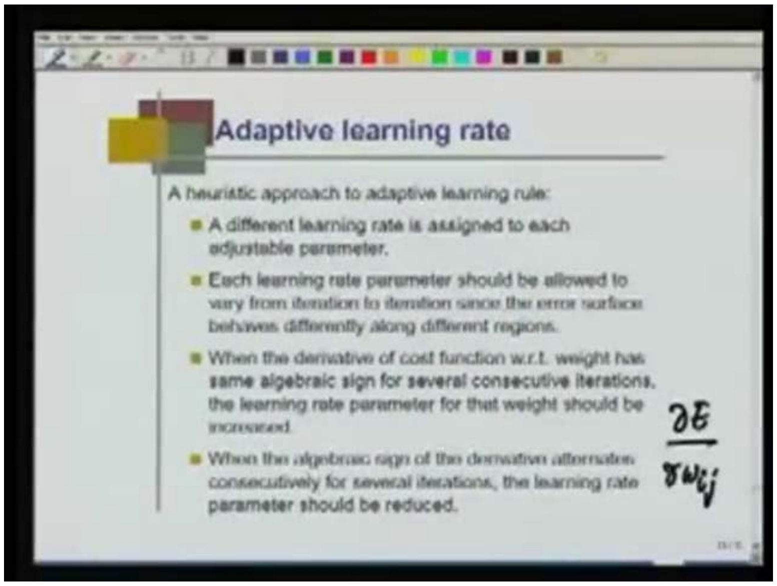
Error Plot

Number of epochs to reach an RMSE of < 0.0005 is ~ 2200, where as for simple gradient descent it is ~ 4200

You can easily see here that when we use the simple gradient descent, we do not use the momentum. The number of iterations is 4200. I take 4200 iterations to converge, that is, my RMS error is less than 0.0005. For reaching the same termination condition (Refer Slide Time:

53:26) momentum, I need only 2200 – almost half of the …. By adding a momentum, I could reduce the number of training by half – this called speed of convergence. The convergence speed is almost doubled using this momentum term. It is not that I can increase the convergence speed by always adding a momentum, but it occurs in some cases; this is heuristic; this is not always true.

(Refer Slide Time: 54:45)



Adaptive learning rate

A heuristic approach to adaptive learning rule:

- A different learning rate is assigned to each adjustable parameter.
- Each learning rate parameter should be allowed to vary from iteration to iteration since the error surface behaves differently along different regions.
- When the derivative of cost function w.r.t. weight has same algebraic sign for several consecutive iterations, the learning rate parameter for that weight should be increased.
- When the algebraic sign of the derivative alternates consecutively for several iterations, the learning rate parameter should be reduced.

$\frac{\partial E}{\partial w_{ij}}$

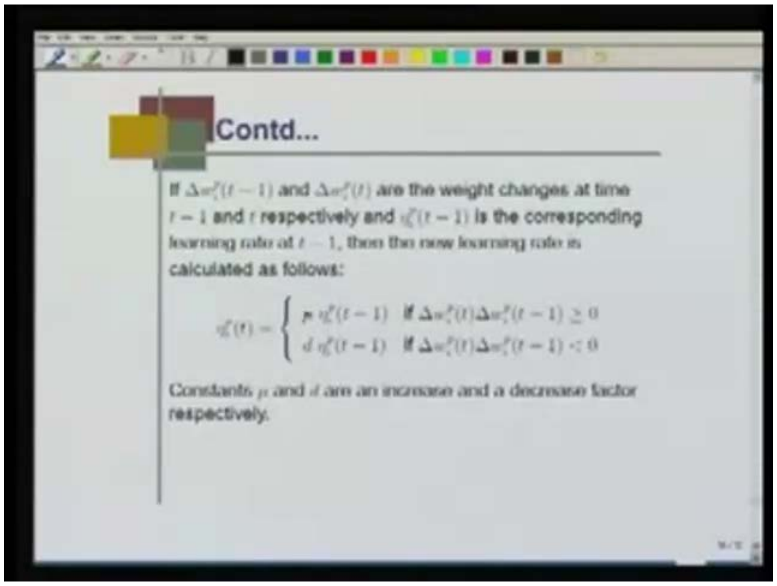
Similarly, for adaptive learning rate, we will have a special class. For today, I will just give you a simple note on how we implement adaptive learning rate in a heuristic manner - a heuristic approach to adaptive learning rule. You see that in back propagation, eta is a fixed value. Once eta is fixed, most likely, you will be in local minima, but if I vary this eta in a very intelligent manner in such a way that it can avoid the local minimum, this is called adaptive learning rate. Can I do that?

Here is a heuristic approach for which we do not have a theoretical basis. Based on certain intuition, we derive this algorithm. What this particular heuristic algorithm is that a different learning rate is assigned to each adjustable parameter; that is, every typical w_{ij} is associated with a different eta. Each learning rate parameter should be allowed to vary from iteration to iteration. When the derivative of the cost function with respect to weight has the same algebraic sign for several consecutive iterations, the learning rate parameter for that weight should be increased.

When the algebraic sign of the derivative alternates consecutively for several iterations, the learning rate parameter should be reduced.

What I am talking about now is that we always compute del E by del w_{ij}, where w_{ij} is the typical weight. If this derivative is constantly positive, then for that particular w_{ij}, I increase the eta. If this derivative alternates, that is, sometimes positive, then negative, positive, negative, positive, negative, then what I do is I decrease this weight; this is the heuristic we introduce.

(Refer Slide Time: 56:08)



Contd...

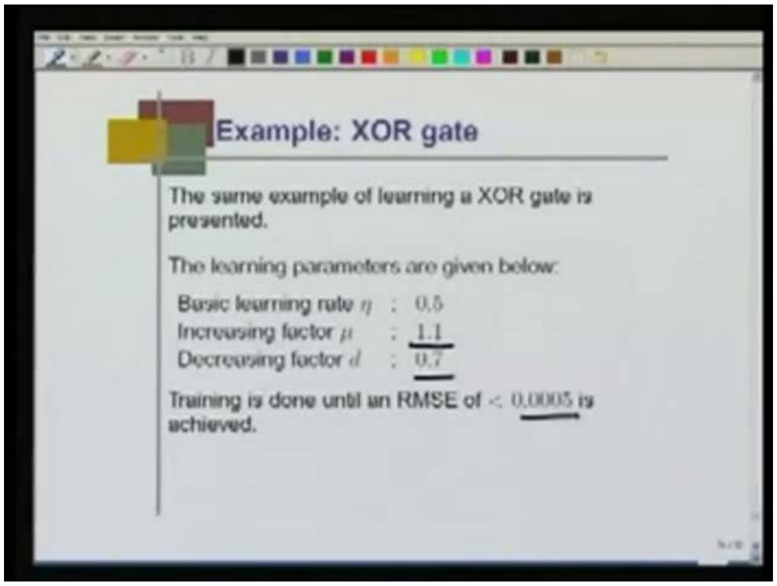
If $\Delta w_i^p(t-1)$ and $\Delta w_i^p(t)$ are the weight changes at time $t-1$ and t respectively and $\eta_i^p(t-1)$ is the corresponding learning rate at $t-1$, then the new learning rate is calculated as follows:

$$\eta_i^p(t) = \begin{cases} \mu\,\eta_i^p(t-1) & \text{if } \Delta w_i^p(t)\Delta w_i^p(t-1) \geq 0 \\ d\,\eta_i^p(t-1) & \text{if } \Delta w_i^p(t)\Delta w_i^p(t-1) < 0 \end{cases}$$

Constants μ and d are an increase and a decrease factor respectively.

Based on that heuristic, this is my adaptive learning rate: eta_i p t is equal to mu eta_i p t minus 1 and d eta_i p t minus 1, where mu and d are the increase and decrease factor, respectively. I either increase this by u or decrease the previous eta by d. This is a multiplication factor. So I either increase it or decrease it.

(Refer Slide Time: 56:51)



Example: XOR gate

The same example of learning a XOR gate is presented.

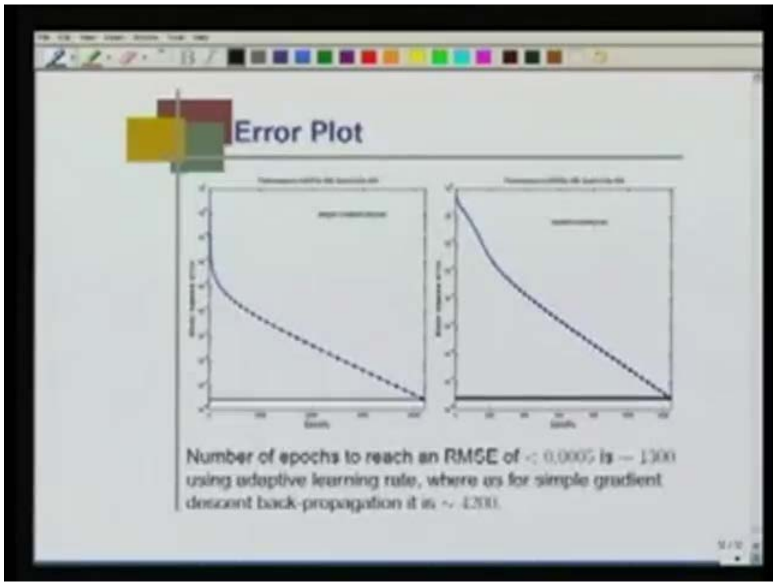
The learning parameters are given below:

Basic learning rate η	:	0.5
Increasing factor μ	:	$\underline{1.1}$
Decreasing factor d	:	$\underline{0.7}$

Training is done until an RMSE of < $\underline{0.0005}$ is achieved.

You see that the increase factor means it has to be greater than 1 and decrease factor means it has to be lesser than 1. You have taken mu to be 1.1, d to be 0.7. We have taken the same XOR function. Now, using the adaptive learning rate, training is done until the root mean square error is less than 0.0005 for the XOR gate.

(Refer Slide Time: 57:26)

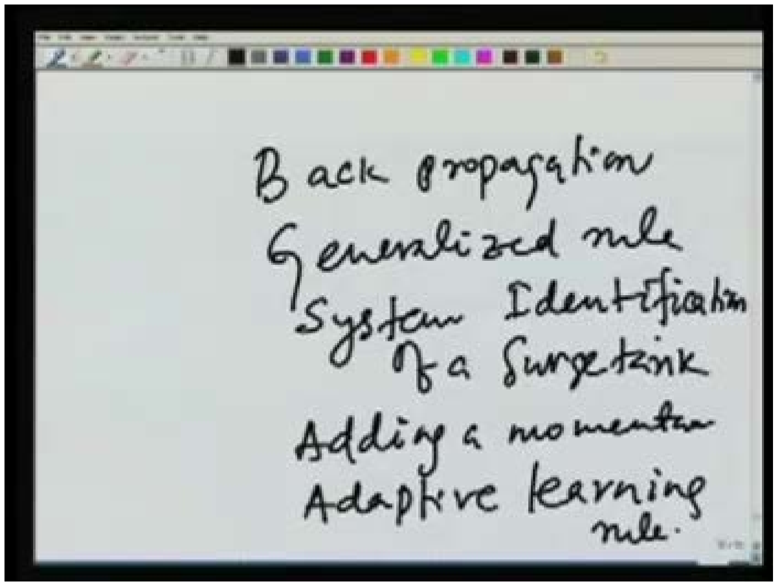


Error Plot

Number of epochs to reach an RMSE of < 0.0005 is ~ 1300 using adaptive learning rate, where as for simple gradient descent back-propagation it is ~ 4200.

We finally see that without adaptive rate, as we have already seen, the simple gradient descent gives you 4200 number of iterations or training samples – I have to update the weights that many times; I have to update the weights 4200 times, whereas using the heuristic adaptive learning rule, we could reduce that to 1300 for the same root mean square error, reaching the same root mean square error.

(Refer Slide Time: 58:16)



Back propagation
Generalized rule
System Identification
of a Surge tank
Adding a momentum
Adaptive learning
rule.

Finally, to conclude, I would say what we discussed today. We reviewed what back propagation is, we talked about the generalized rule and then we talked about system identification of a surge tank. Then, we talked about adding a momentum – this is to increase the convergence speed as well as avoid local minimum. We also talked about a heuristic adaptive learning rule. We will take a special class on adaptive learning rule in detail - comprehensively how we can compute this adaptive learning rule having a theoretical basis; probably after two or three classes, but this is the summary of this class. Thank you very much.