Intelligent Systems and Control

Prof. Laxmidhar Behera

Department of Electrical Engineering

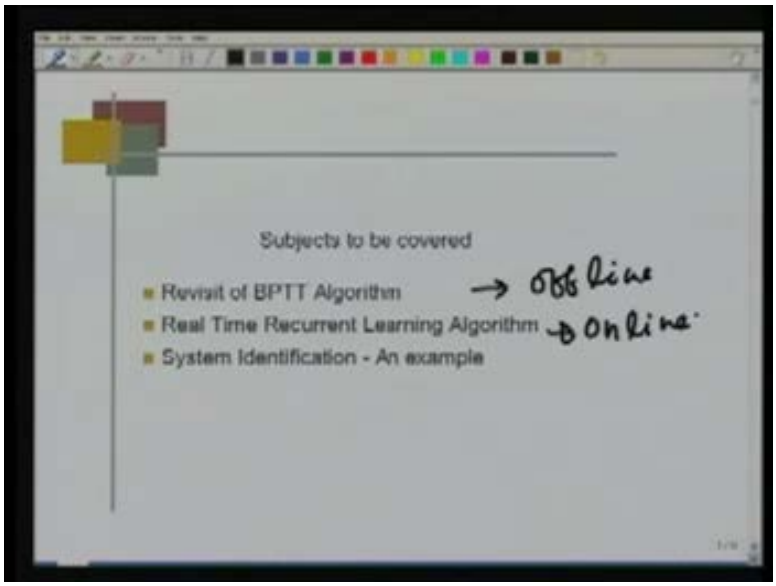
Indian Institute of Technology, Kanpur

Module – 1 Lecture – 10

Recurrent Networks: Real Time Recurrent Learning

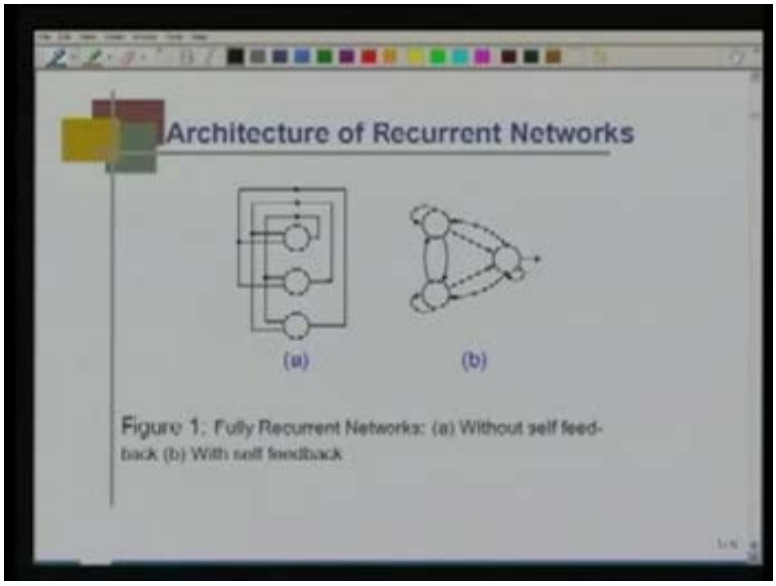
This is the tenth lecture of the first module. We will be discussing today the second part of recurrent networks.

(Refer Slide Time: 00:29)



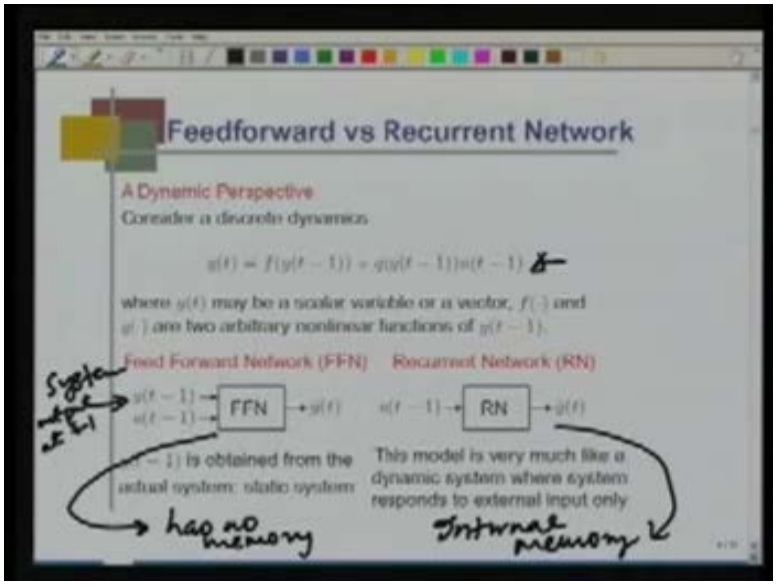
The subjects to be covered in this section … we will revisit back propagation through time algorithm that we discussed in the last class and then we will introduce a real-time recurrent learning algorithm. The previous algorithm that we introduced in the last class was offline and this is online. This was an offline in the sense that we collect the data a priori from a system and then model the system, whereas we collect data online from the system, that is, as we collect the data, we learn the model of the system. Then, we will see an example of system identification using RTRL and we will compare with the previous result, that is, simple back propagation, back propagation through time and RTRL.

(Refer Slide Time: 01:46)



As I said, recurrent network means you have all these feedbacks. These are your individual neurons and the neuron output goes as feedback to all these when they become input to the neuron. This is a feedback connection. Each neuron is connected and all the neuronal responses are fed back to the input. This is called fully connected recurrent network. Similarly, here also, it is almost the same structure. If you look at this, the difference between this and this is that we have a self-feedback here. That means I take from here and put it here, but in this case, I do not use this self-feedback. This one (Refer Slide Time: 02:52) goes only to this neuron and this neuron. This is a fully recurrent network, both are, but this is with self-feedback and this is without self-feedback, that is, each neuron is connected to each other in feedback.

(Refer Slide Time: 05:26)



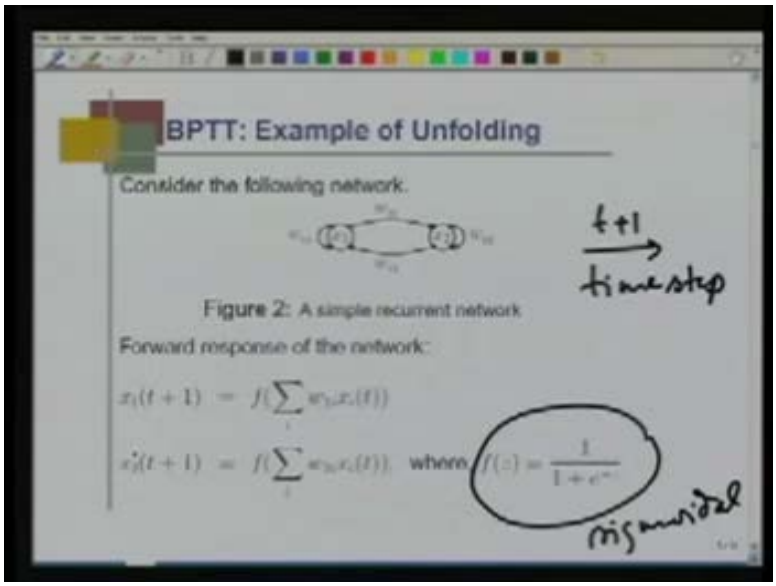
We also discussed what is feed-forward versus recurrent network. If you look at a feed-forward network, let us say, a discrete dynamic, a mathematical form of a dynamic is y t is a function of y t minus 1 and f is a nonlinear function and g is g of y t minus 1 (this is another nonlinear function of y t minus 1) into u t minus 1. This is a specific type of nonlinear dynamics. Obviously, y t is a function of y t minus 1 as well u t minus 1. In a feed-forward network, if I model this dynamic, how do I do it?

I have a feed-forward network. This feed-forward network can be a back propagation network or a radial basis function network and the output is computed while inputs are the previous actual output of the system and the input to the system, that is, u t minus 1. This is the system output (Refer Slide Time: 04:27) and not the network output. This is system output at instant t minus 1. From the system, I take the output, which is y t minus 1, and what is input to the system is u t minus 1. Based on that, I predict what the system output at t is.

This is the predictive model using a feed-forward network for this type of dynamics, but if I want to capture this dynamic using a recurrent network, the structure is that I only give input and I see what is at y t. What you see is that this particular network has no memory. Hence, this is a static network, whereas this has internal memory and so this is very much like a dynamic system; in a dynamic system, we give input and see the output. A system is dynamic when it has internal

memory. A recurrent network is more like a dynamic system, whereas a feed-forward network is a static network simply constructed for a function approximation.

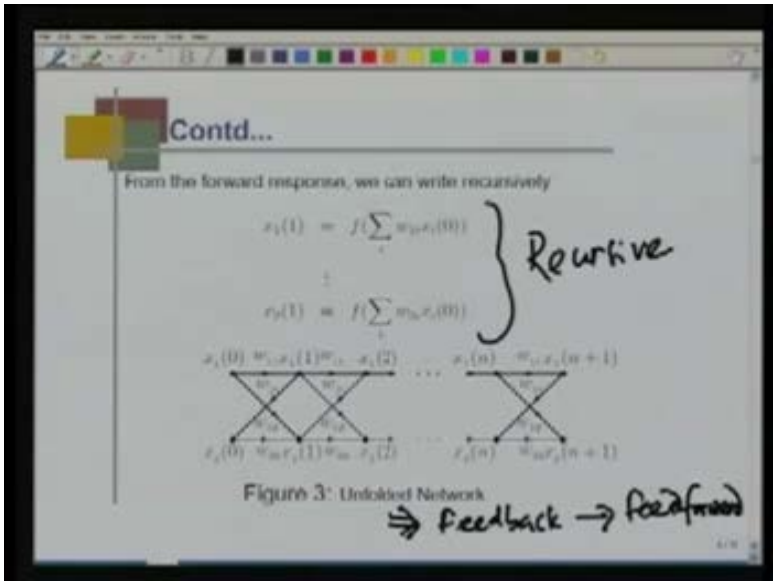
(Refer Slide Time: 06:25)



We discussed the BPTT example last time. We also considered this particular example. We have two neurons connected to each other. This is the connection from first neuron to second neuron; this is connection from second neuron to first neuron with self-feedback w_{11}, w_{22}. If you look at this particular thing, we also discussed last time that the output, the response x_1 t plus 1…. This t is not time; t plus 1 is the time step; t is a numerical integer – 0, 1, 2, 3, 4, 5, and so on. So, this is time step.

So x_1 t plus 1. In discrete dynamics, we do not represent t in terms of absolute time; it is in terms of time step. So, x_1 t plus 1 is a function of w_{1i} x_i t. You can easily see x_1 t plus 1, x_1 t plus 1 here (Refer Slide Time: 07:28) is a function of w_{12} into x_2 t plus w_{11} into x_1 t. So, w_{11} x_1 t, w_{12} x_2 t, and a function. Normally, we use a sigmoidal activation function. f is sigmoidal activation function. Here it is (Refer Slide Time: 07:59). Similarly, in this case, x_2 t plus 1 is a function of w_{21} and w_{22}. w_{21} is multiplied with the corresponding input signal x_1 t and w_{22} is multiplied with corresponding input signal x_2 t. This is our forward response of the network.

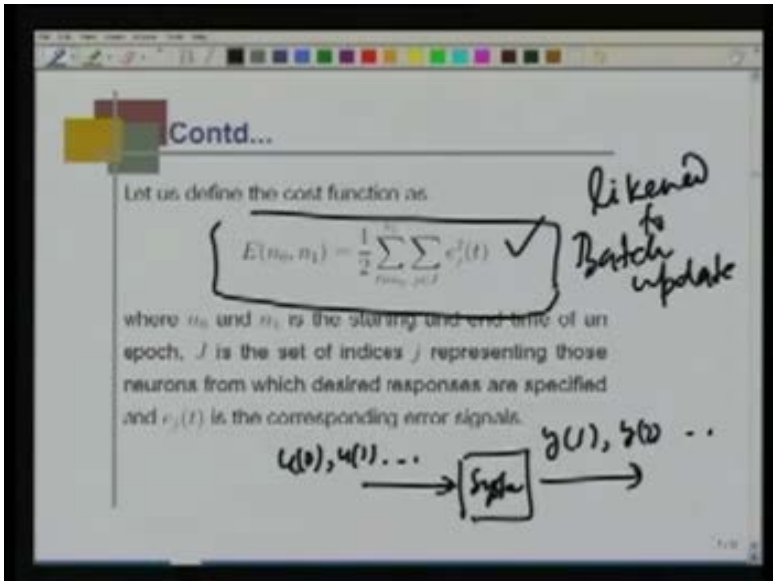
(Refer Slide Time: 0826)



In the previous one (Refer Slide Time: 08:32), you see that I can put in the recursion. What is the meaning of recursion? I put t = 0, so x_{11} in terms of 0, x_{12} in terms of 1. Similarly, x_{13} is in terms of 2. This is a recursive relation. This is a recursive relation (Refer Slide Time: 09:00). What I am trying to do is that we represent the response of the network at specific time – t = 1 in terms of t = 0. Similarly, the response at t = 2 in terms of t = 1 and we unfold the network according to the application. So n plus 1 time step we go and expand the network.

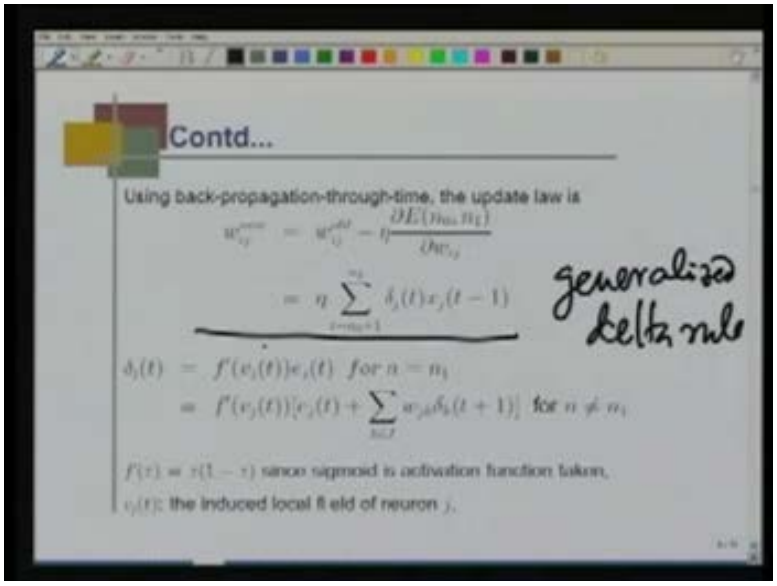
The objective here is that we convert a feedback network to feed-forward network. Unfolding network means convert a feedback network to a feed-forward network. That is the objective of unfolding.

(Refer Slide Time: 10:12)



Then, we define as usual a cost function, which is the instantaneous error at an individual node and sum over all the nodes and then over time steps. As you see, our cost function in this particular case is a kind of batch update cost function, because the back propagation through time is an offline technique, that is, I give to the system a sequence of inputs. If this is my system, I give a sequence of inputs, that is, u 0, u 1, and so on. Then, I observe the sequence of outputs, y 1, y 2, and so on. Then, I collect this data. Over the total batch, for example, I have 200 sequences of input, so we have 200 sequences of input/output data. Over those data (Refer Slide Time: 11:24), we define the cost function. This is a kind of batch. I would not say batch update, but it can be likened to a batch update, because that is how the back propagation network works.

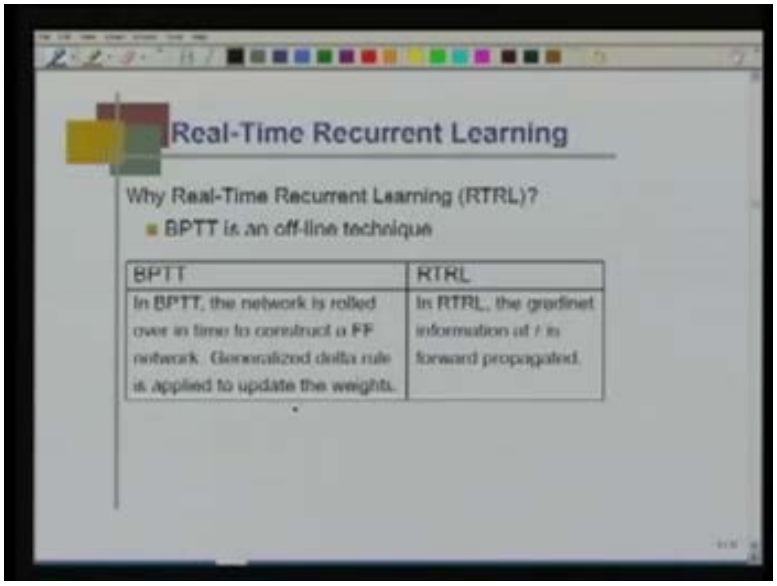
(Refer Slide Time: 11:43)



We showed how after you convert a feedback network to a feed-forward network, you simply apply the back propagation generalized delta rule. If I apply the generalized delta rule, we found that this is the generalized delta rule. Applying the generalized delta rule…. I find out the error here (Refer Slide Time: 12:21), I back propagate this error to this and update these weights. I found out what should be the delta w in this layer. Then, the error back propagated through this weight is added with the error at the instant n at both the nodes. Then, again, the combined error is back propagated until I reach the last layer here. That is what we do in back propagation through time.

That is what you are seeing here (Refer Slide Time: 13:00). Error computed from one layer is back propagated to the preceding layer. This is the n th layer. From the n th layer, we go to n minus 1 layer and then we go subsequently until we find the delta w at every layer and then we sum all these delta ws and then update the weight like batch update. We have already discussed this. Today, we will discuss real-time recurrent learning. What is the difference, real-time recurrent learning?

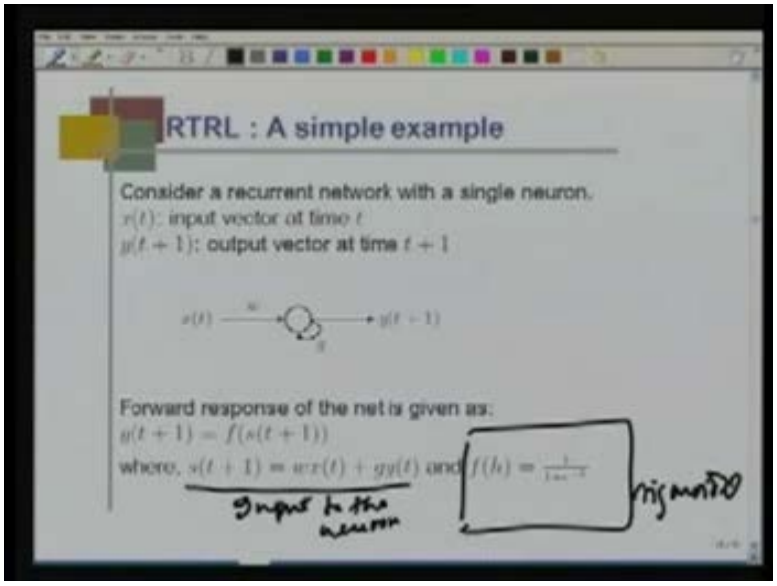
(Refer Slide Time: 13:41)



It is real time and that means I do not have to collect data over time from the system and then learn the model. No, I learn the model as and when the data comes; most of our control problems must be dealt online. In that sense, real-time recurrent learning has better utility than back propagation through time. As usual, as I have already said, in back propagation through time, the network is rolled over in time to construct a feed-forward network. The generalized delta rule is applied to update the weights. That is what we discussed just now.

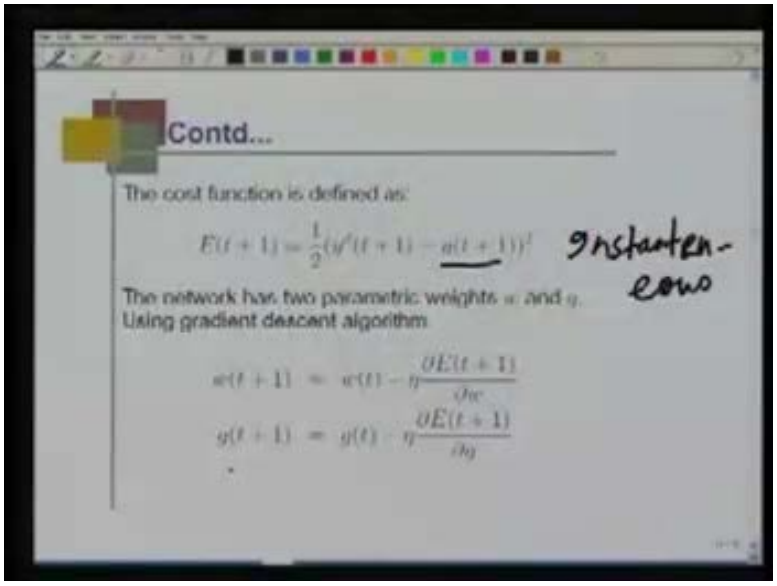
In contrast, RTRL, that is, real-time recurrent learning, the gradient information at t is forward propagated, that is, I do not construct a feed-forward network; rather, at t constant, whatever the gradient information I have, I forward propagate that gradient information to time step t plus 1. The gradient information at t plus 1 is forward propagated to compute the gradient information at t plus 2 and so on. This is the difference. The gradient information in back propagation through time is backward propagated through time, whereas gradient information in case of RTRL is forward propagated in time. This is the major difference. Of course, RTRL is a real-time or online technique, whereas this is an offline technique.

(Refer Slide Time: 15:39)



Now, we will take a very simple example. We also took a simple example for back propagation through time. We have a single neuron, I have input x t, the output is y t plus 1, we have two weights w and g and this is a unit delay. The input to the neuron is this (Refer Slide Time: 16:03). This is the input to the neuron. That is w x t, which is input, plus g y t, where y t is the previous output. That means this network has an internal one-step memory. x t plus 1 is w x t plus g y t. Then, this neuron has a nonlinear activation function. The output of the neuron, which is y t plus 1, is a function of s t plus 1, where f h is a sigmoidal activation function. These are the normal structures or normal ways to compute the response of a recurrent network that we have already seen.

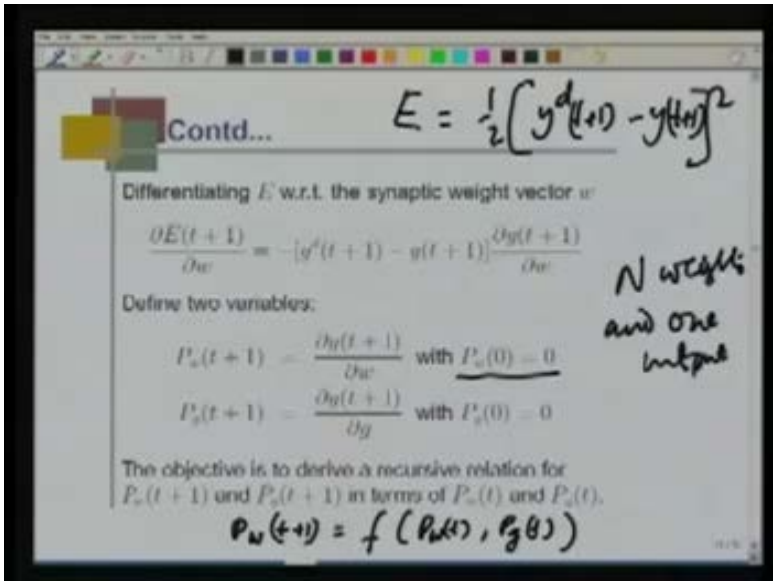
(Refer Slide Time: 16:57)



Now, you see that we will not compute a cost function like the batch update that we do in back propagation network or back propagation through time. This is instantaneous. The cost function is simply a quadratic function of the instantaneous error, that is, half (y d t plus 1 minus y t plus 1) square. This is desired and this is the actual output of the network (Refer Slide Time: 17:44). In this network that we showed, we have only two weights w and g. Using gradient descent algorithm, how I write this weight is w t plus 1 is equal to w t minus eta del E t plus 1 upon del w.

In a simple back propagation network, we did not put t plus 1 as error. The error is not a function of t plus 1, but in recurrent network, it is a function of the time step. Why? It is because it has internal memory and that is why these are dynamic networks. Similarly, g t plus 1 is g t minus eta del E t plus 1 upon del g. The problem here is that for a recurrent network, how do we compute the gradient error function with respect to weight? That is what we will see now.

(Refer Slide Time: 18:42)



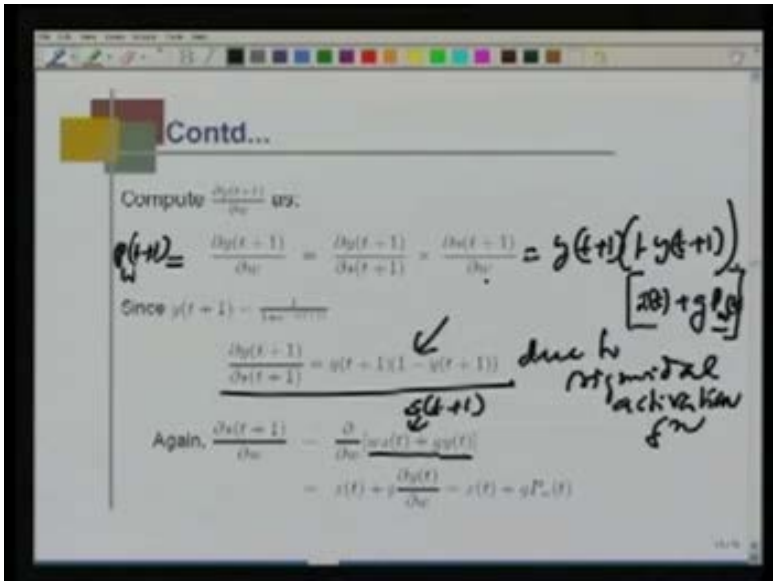
If you differentiate E with respect to w, you have as usual y d t plus 1 minus y t plus 1 and minus sign will come because you again differentiate y t plus 1 with respect to w. There is a minus sign (Refer Slide Time: 19:02) and this minus comes due to that. Just for those who did not remember what is the meaning of E, it is simply half of y d t plus 1 minus y t plus 1 whole square. If you differentiate E with respect to w, you have this term coming out. Then, you differentiate this term with respect to w and so, minus comes out there.

Now, we will define two terms, which are actually gradient terms. P_w t plus 1 is the gradient term at t plus 1 and this term is defined as del y t plus 1, not del E. Please note we are not defining as del E but it is del y – response of the unit or neuron. So, it is del y t plus 1 upon del w, the partial derivative. Similarly, P_g t plus 1, g is another weight…. If I differentiate y t plus 1 with respect to del g, then it is P_g with the assumption that P_w 0 is 0. What is P_w 0? P_w 0 is del y 0 upon del w. Similarly, P_g 0 is del y 0 upon del g. So, the partial derivative at time instant 0, that is, the initial condition, is 0.

The objective is to derive a recursive relation between P_w t plus 1 and P_w t and P_g t and similarly, P_g t plus 1 in terms of P_w t P_g t. This is theme of real-time recurrent learning (RTRL), that is, I want to find a relationship P_w t plus 1 in terms of P_w t and P_g t. So, I have two gradient terms. You must know that if I have N weights and one output, then the gradient term at every

instant should be N. More number of responses, based on that, we will have more gradient information at every time. I will discuss this that, but in this case, here (Refer Slide Time: 22:11), I have only one output, two weights, and only one neuron, which has only one response. Obviously, at every instant, the number of gradient terms will be 2. That is what we will do now. We will relate P_w t plus 1 and we will try to compute the gradient term at t plus 1 based on the gradient term in the previous instant at t.

(Refer Slide Time: 22:45)



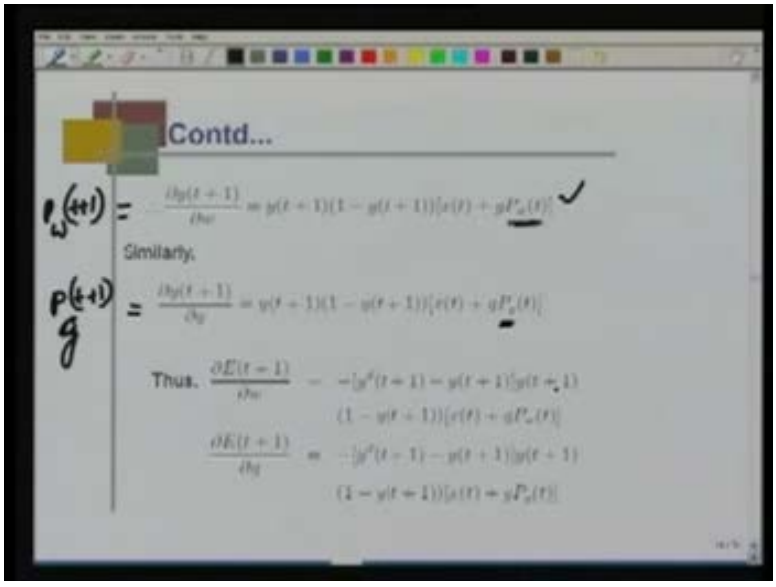
Let us now compute. This is P_w. I can write here that this is actually P_w t plus 1 by definition. So, P_w t plus 1, which is this particular term, is y…. I can differentiate y with respect to s t plus 1 because y t plus 1 is f of s t plus 1 and then del s t plus 1 upon del w. Since y t is a sigmoidal activation function of its input s t plus 1, obviously, we know this particular term (Refer Slide Time: 23:25) is simply y t plus 1 into (1 minus y t plus 1) due to sigmoidal activation function. We can verify that.

When this is the structure, you differentiate this with respect to s t plus 1, then we will have this particular term (Refer Slide Time: 23:50). Similarly, del s t plus 1 by del w…. So, s t plus 1 is this; s t plus 1 is w x t plus g y t and I have to differentiate this with respect to w. By definition, when I differentiate this particular term with respect to w, I get first x t and because x t is an input, I cannot differentiate input with respect to w because the input is not a function of w and

output is a function of w or any neuronal response is a function of input. Try to remember that this x t is not a function of w. Hence, when I differentiate this term, I only get x t with respect to w. Similarly, in g y t, g is simply a weight and it is not a function of w. Simply, it is g del y t upon del w by differentiating this (Refer Slide Time: 24:51) and by definition, the first term is x t plus g and by definition, this one is P_w t, the gradient information at time t – one of the gradient information.

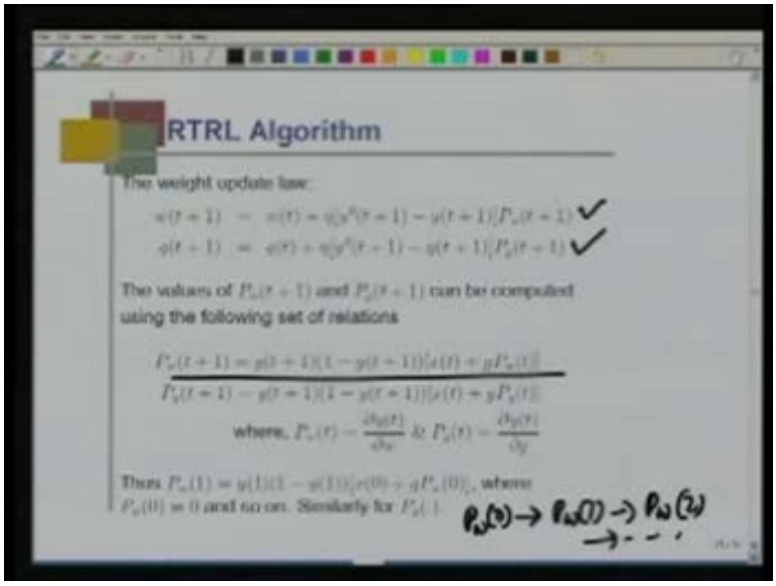
What you get is P_w t plus 1 is finally y t plus 1 into 1 minus y t plus 1 into x t plus g P_w t. What you are seeing is that P_w t plus 1 is written in terms of P_w t. So, the gradient information at t plus 1 is a function of gradient information P_w t. If I have gradient information at t, I can easily compute gradient information at P_w t plus 1. If I know the gradient information at t, then I can compute what the gradient information at t plus 1 is. This is the basic theme of real-time recurrent network. Once you understand this, you can derive any complex network.

(Refer Slide Time: 26:16)



This is what we derived. Similarly, you can see the other one is also in the same way, which is this one in this side (Refer Slide Time: 26:22). This is P_g t plus 1 and this one is actually P_w t plus 1. P_w t plus 1 is a function of P_w t and P_g t plus 1 is a function of P_g t. Finally, we can write del E t plus 1 by del w is simply this. This is the error term and this particular term is the gradient information at t plus 1. Now, it is a very simple relationship.

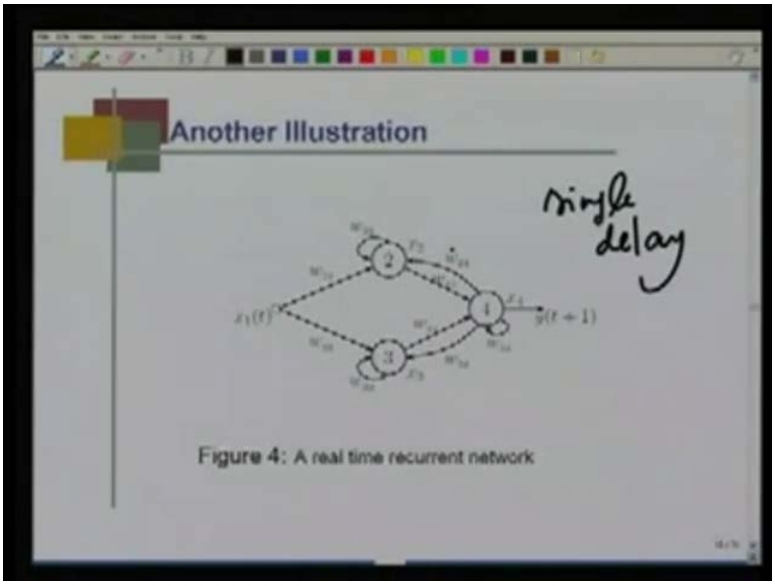
(Refer Slide Time: 27:04)



Now, weight update law. This is the final weight update law (Refer Slide Time: 27:11). What you saw is that in this weight update law, w t plus 1 is simply this previous weight into the instantaneous error t plus 1 into learning rate and gradient information at t plus 1. How do I compute this gradient information P_w t plus 1? Using this recursive relationship (Refer Slide Time: 27:36), that is, P_w t plus 1 is computed using the gradient information that is already computed at t. That is P_w t into g plus x t, input at t into this particular information.

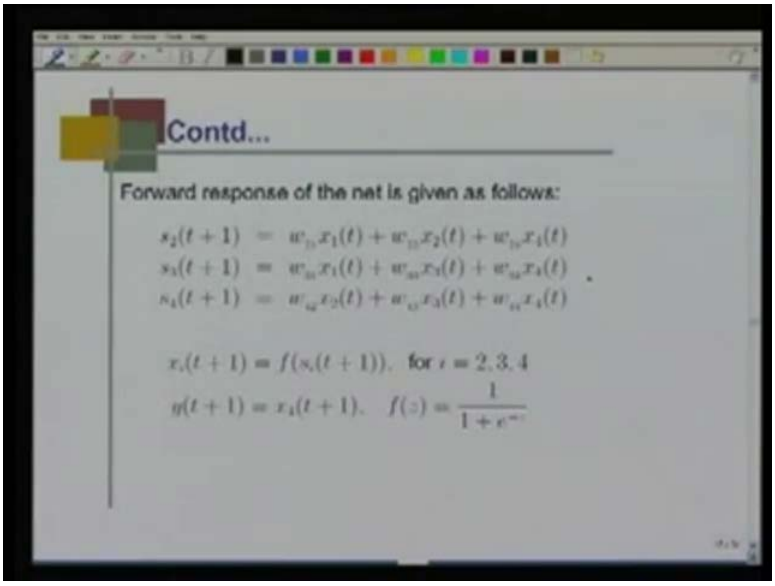
Similarly, P_g t plus 1 is also computed here: the gradient information at t plus 1 – second gradient information using the previous. How do we go about it? In the beginning, first, at 0 instant, we assume P_w 0 and P_g 0 to be 0. Once we compute that, then we can easily compute what is P_w 1. Once P_w 1 is computed, (Refer Slide Time: 28:28) P_w 2. You see that if I put t = 0, then P_w 1 is y 1 into 1 minus y 1 into x t, which is x 0, plus g P_w 0 and this is 0 and so on. Once I have P 1, I can compute P 2, with P 2, P 3 and it goes on like this. I start from P 0 and then compute P. Let me not confuse you again like back propagation. I will put in terms of forward propagation. So, I have P_w 0, this is 0. From there, I compute P_w 1and from there, I compute P_w 2 and so on. This is called forward propagation of gradient descent algorithm.

(Refer Slide Time: 29:25)



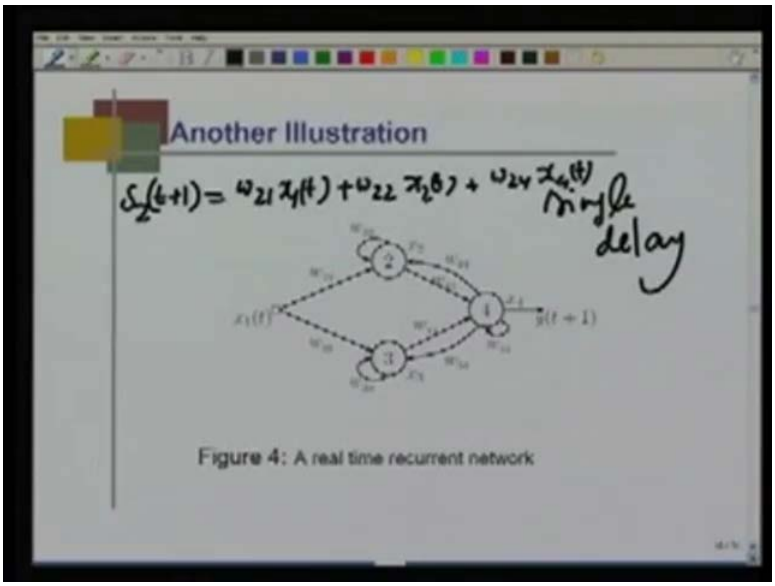
We are now ready to discuss a little more complex network, which was a simple last one, a single neural network. Now, I have three neurons 1, 2, 3. Are you aware of how we put the network weights? You see that this is the second neuron. The weights coming to this neuron is all initialized w_2 and from where is it coming? From 2. This is w_2. From where is it coming? From 1, so this is w_{21}. Similarly, this is w_2 and coming from 4, w_{24}. There is no connection between 2 and 3. That means this is partially connected and not fully connected. Fully connected means there should be at least a connection from here to here. Similarly, w_4 is connected from 2 and 3 and not from 1. These are the conventions. You see that w_{43} is a connection weight to unit 4, neuron 4 from unit 3; similarly, w_{34} is a connection weight from 4 to 3. Of course, w_{33}, w_{22} and w_{44} are all self-feedback. They are connected self-feedback. We assume that all these connections have single delay.

(Refer Slide Time: 31:15)



Based on that assumption, we will write what is the forward response of the network. s_2, s_3 and s_4 are the total input sum to the neurons 2, 3, and 4, respectively (Refer Slide Time: 31:20). So s_2 t plus 1 is w_{21} x_1 t plus w_{22} x_2 t plus w_{24} x_4 t. You can see that here (Refer Slide Time: 31:38).

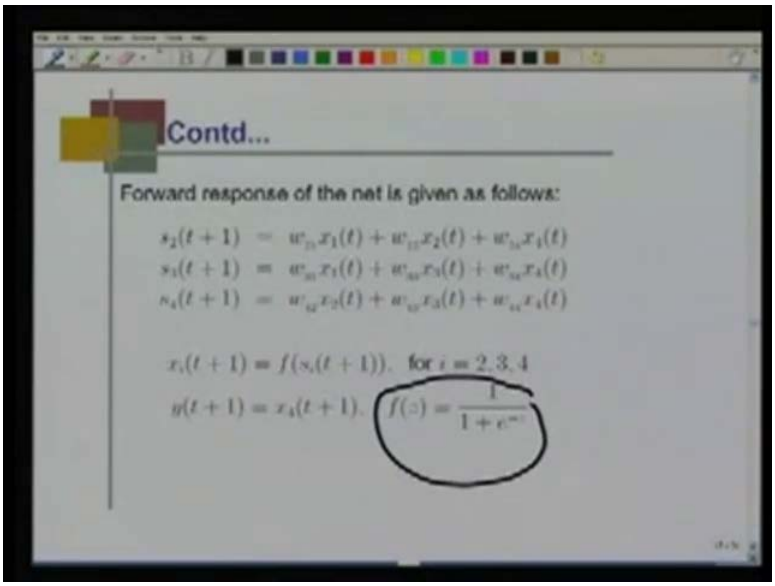
(Refer Slide Time: 31:39)



The input to the network 2 I can say is s_2 t plus 1, which is the input to the neuron 2, is w_{21} into x_1 t plus w_{22} into x_2 t plus w_{24} x_4 t. There are totally three input signals to this neuron 2. This is

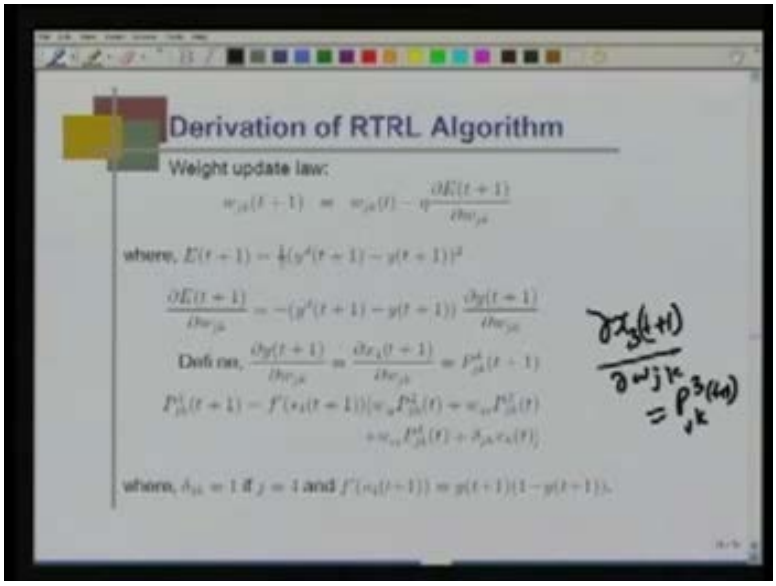
the summed input to the neuron. Based on sigmoidal activity, we compute x_2 t plus 1 is f of s_2 t plus 1 (Refer Slide Time: 32:47). For the forward response of the network, you just compute what is s_2 t plus 1, s_3 t plus 1, s_4 t plus 1. After computing this sum, the sum of the total input to the neuron, then the final response is the sigmoidal activation of the total sum input. Finally, the response of the network you will get here is y t plus 1, which is x_4 t plus 1 (Refer Slide Time: 33:20).

(Refer Slide Time: 33:22)



That is what we write here y t plus 1 is x_4 t plus 1 All activation functions are sigmoidal f.

(Refer Slide Time: 33:34)



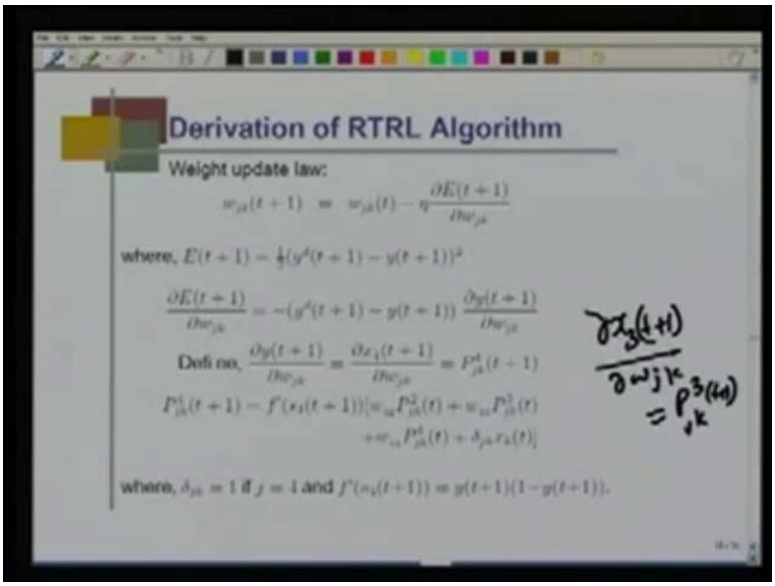
Derivation of sigmoidal activation function. The weight update law is as usual a gradient descent law, but now my cost function is a time function that is dynamic, because E t plus 1, which is y t plus 1…. So, y t plus 1 is a function of y t as well as the other neuronal internal states. That makes the difference. E t plus 1 is the instantaneous quality function, which is half into (y d into t plus 1 minus y t plus 1) square. This is again instantaneous error because this is real-time recurrent learning, that is output. I want to now compute E t plus 1 upon del w_{jk}. What is w_{jk}? It can be anything.

How many weights do we have here? If you see here (Refer Slide Time: 34:58), we have a total of nine weights – 1, 2, 3, 4, 5, 6, 7, 8, 9. Totally, there are nine weights. Obviously, with respect to x_4 or y, we have nine gradient information at any instant. Similarly, with respect to 2, here also, there is gradient information; with respect to x_2, we have total connection here and we can compute all the weights actually, because x_2 can be a function of all the weights, because that is the meaning of feedback. x_2 t plus 1 is a function of all the weights. This is very important to understand.

del E t plus 1 del w_{jk} is obviously…. If I want to differentiate this with respect to any typical weight w_{jk}, we get here, this particular term (Refer Slide Time: 36:06), error. When I differentiate this with respect to w, the negative comes out and we will define this partial

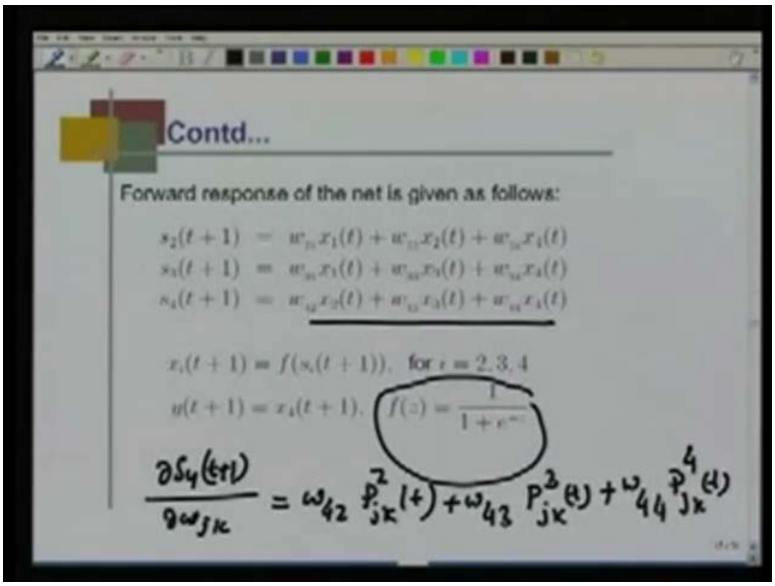
derivative del y t plus 1 upon del w_{jk}, this partial derivative. Instead of y, we will write x_4, because that is the convention we will select that for all active neurons, the output is x. So, x_1, x_2, x_3, x_4, as many neurons as there, that many subscripts will be there for x. y is x_4 t plus 1 and we will define this particular function as P_{jk} 4, which means the differentiation of x_4 t plus 1 with respect to w_{jk} is P_{jk} 4 to the power of t plus 1. It is the same thing when I compute at t, it will be t, t minus 1 will be t minus 1, but the way to write this is P_{jk} 4.

(Refer Slide Time: 37:19)



Similarly, if I want to write del x_3 t plus 1 by del w_{jk} will be P_{jk} 3 t plus 1. This is where we define the gradient information. As usual, we showed in the simple recurrent network with one neuron. Similarly, if you go on computing this particular thing, del y t plus 1 upon del w_{jk}, then you will differentiate y t plus 1 with respect to s 4 t plus 1. This is a function of s 4 t plus 1. If you differentiate… and then s_4 t plus 1 with respect to w_{jk} if you differentiate…. If we want differentiate this x_4 with respect to w_{jk}, you can easily see that previously, s_4 is this particular term.

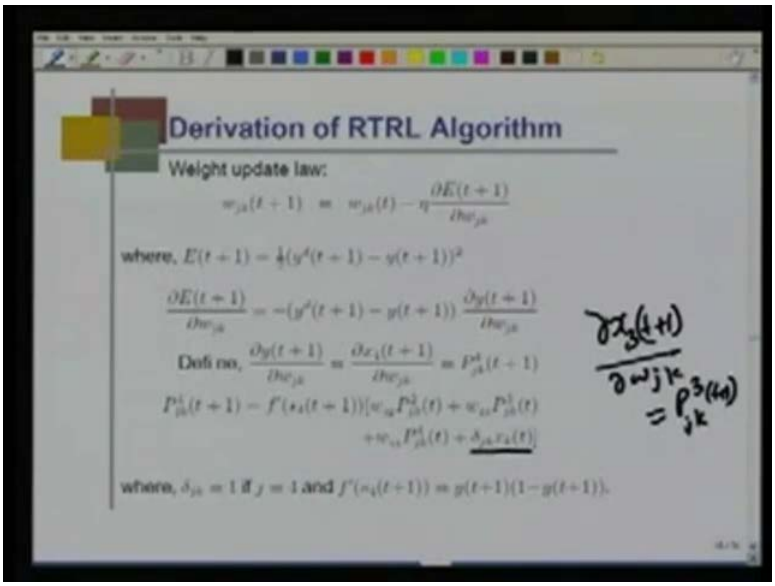
(Refer Slide Time: 38:32)



If I differentiate del s_4 t plus 1 by del w_{jk}, you can easily see I am differentiating this particular term with respect to del w_{jk}. So, this is a constant term w_{42} and you differentiate x_2 t with respect to w_{jk}. By definition, this is P_{jk} 2 t (2 will come here) t. The second one is w_{43} – a constant term again. Then, you differentiate x_3 t with respect to w_{jk}. Then again, this is P_{jk} 3 t plus and again here, this is a constant w_{44} and P again with respect to 4 and jk t. By definition, we are computing s_4 in terms of gradient information in the previous step.

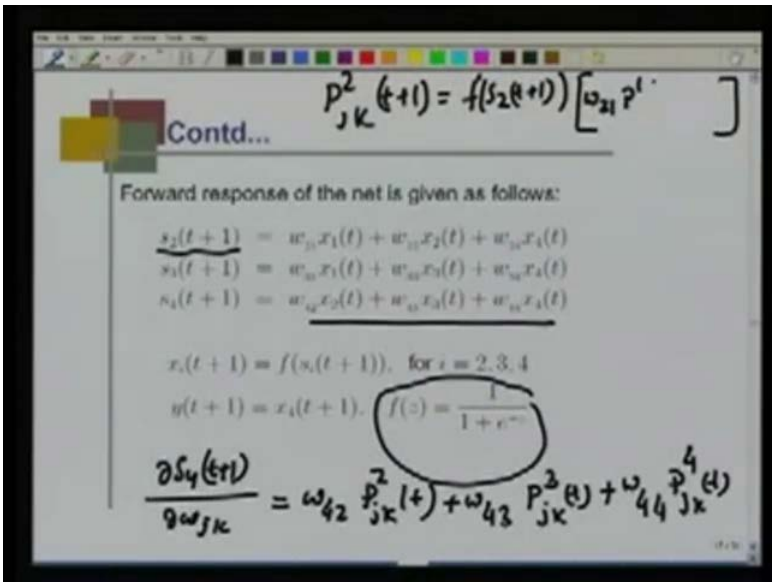
By doing that, you easily see that P_{jk} 4 t plus 1 is this particular term (Refer Slide Time: 39:55) and the extra term that is coming here is…. If you imagine that jk by chance is 42, 43, or 44, it is not actually complete information; I made a mistake. w_{jk} can be either 42 or 43 or 44. If that is the case, then obviously, if it is 42, then differentiation of this will be also x_2 t, simply x_2 t. If jk is 43, then the differentiation of this with respect to w_{jk} besides this term is also x_3 t.

(Refer Slide Time: 40:58)



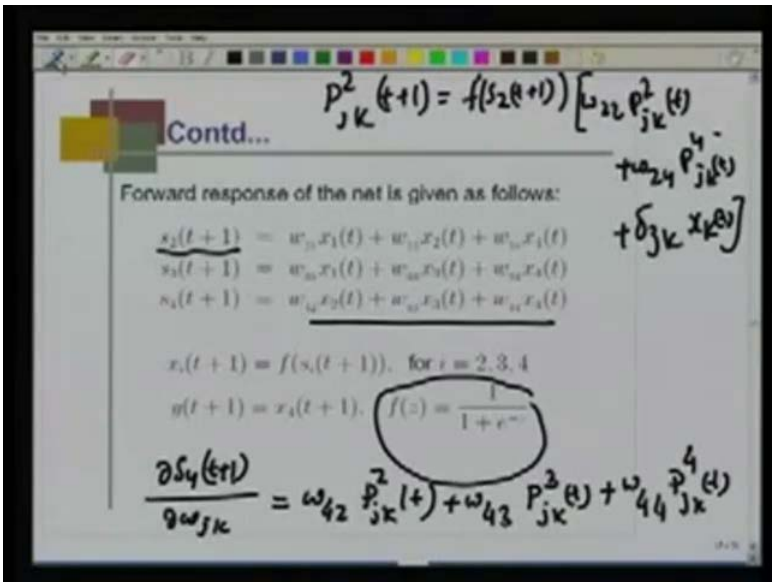
This is what is the last term, that is, delta$_{jk}$ is 1 if j =o 4. Then, this is 1 into x_k, that is, either x_2 or x_3 or x_4. That completes how to compute P_{jk} 4. We learnt how to compute the gradient information, which is Pj_k 4 at t plus 1 with respect to the other gradient information at t, but, what about this? In this particular expression, there is also gradient information – Pj_k 2 t, Pj_k 3 t, Pj_k 4 t. We already derived Pj_k 4 t. We have to also find the gradient information, that is, how do we relate Pj_k 2 t plus 1 with respect to all other gradient information? This also we can always write because this is simply….

(Refer Slide Time: 42:13)



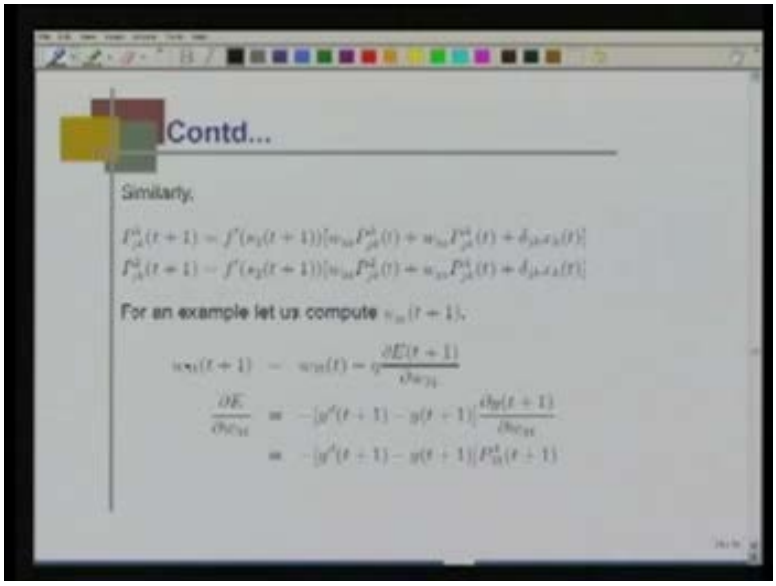
If I go to the previous page and if I write P_{jk} 2 t plus 1 will be f s_2 t plus 1, derivative of that. x 4 t plus 1, x 2 t plus 1 into 1 minus x 2 t plus 1 into 1… into…. You can easily see that I have to differentiate now this particular term (Refer Slide Time: 42:53) with respect to jk. It is the same. What we computed here, this would be simply w_{21} and this will be P 1. You must know that x_1 is an input. So, I cannot actually write this.

(Refer Slide Time: 43:14)



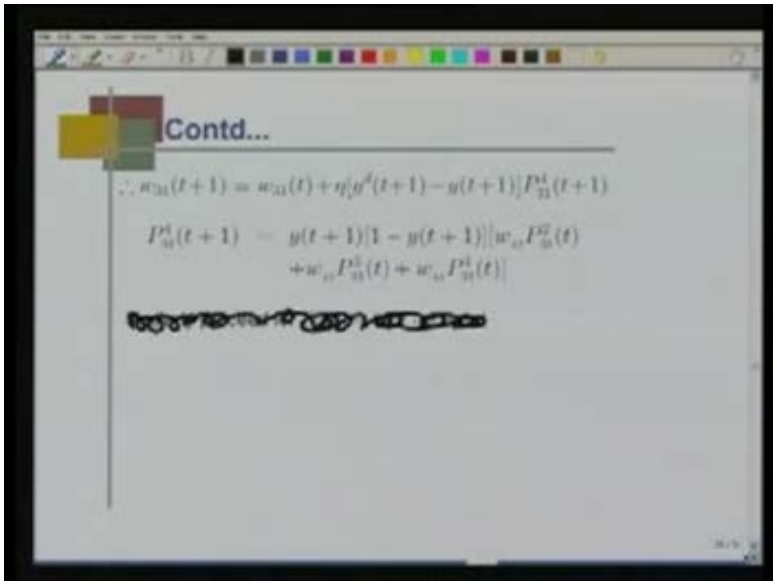
If I differentiate this with respect to w_{jk}, x_1 t is an input. Input is not a function of w_{jk} and so, we will omit that. First, we will start from here. That is w_{22} P_{jk} 2 t plus w_{24} P_{jk} 4 t plus $delta_{jk}$ x_k k. If jk is either 21 or 22 or 24, then the corresponding term x_1 t or x_2 t or x_4 will appear here. This is your P_{jk} 2 t plus 1 in terms of the other gradient information at t. Similarly, as we saw, we can find out the gradient information recursive relation for P_{jk} 3, P_{jk} 4, and so on (Refer Slide Time: 44:27). Once we do that….

(Refer Slide Time: 44:30)



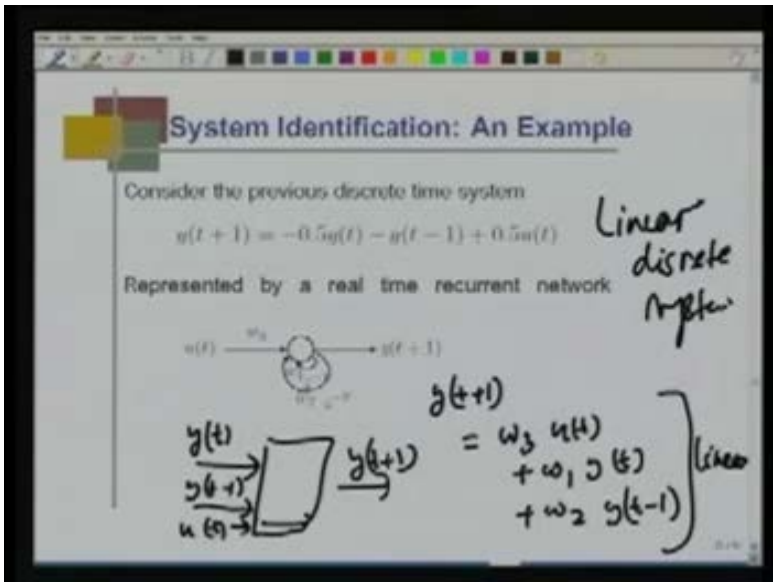
Similarly, we got P_{jk} 3, P_{jk} 2 and then, we do not have to go exclusive. For example, we simply say jk. Instead of jk, if I write 31, how do I compute?

(Refer Slide Time: 44:50)



You can easily see P_{31} 4 t plus 1 was this particular thing.

(Refer Slide Time: 44:57)



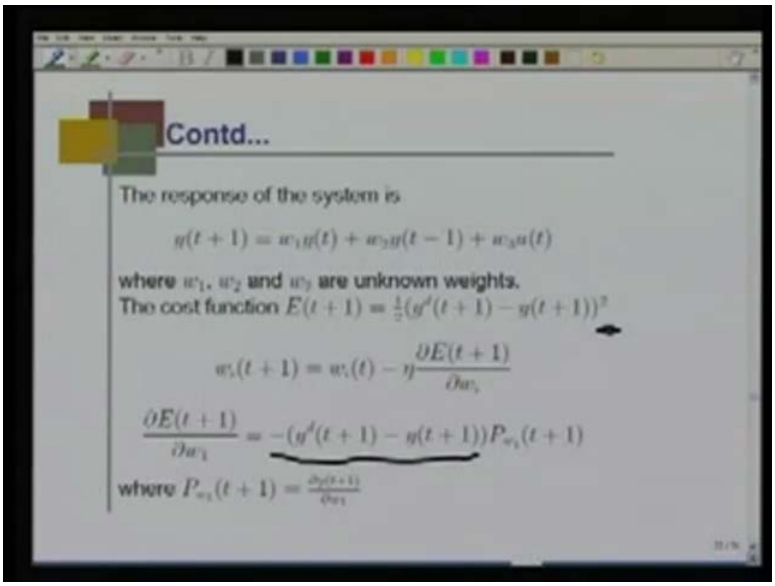
Now, we go to system identification: an example. We have taken this example earlier in a previous class. This is a linear dynamical discrete system. y t plus 1 is minus 0.5 y t minus y t minus 1 plus 0.5 u t, represented by a real-time recurrent network. How do I represent this network? We have already explained. In a feed-forward network, this will be y t from the

system, u t, this is y t minus 1 and one more is u t. We have three inputs and the output is y t plus 1. In contrast, the recurrent network is a memory network because it has its own internal memory, u t is the input and y t plus 1 is the output and it has its own (Refer Slide Time: 46:19). Here, the first one is single delay and second one is double delay. What does it mean?

I can write now y t plus 1 is w_3 u t plus w_1 into this is a single delay, so, w_1 into y t – single delay and double delay is w_2 into y t minus 1. If I match this into this, you can easily see… and because this is a linear discrete system, in this case, y t plus 1 is simply w_3 u t plus w_1 y t plus w_2 y t minus 1. This is linear. This network response is this, the network has this response, linear dynamic. Obviously, I do not know w_3, w_1 and w_2 of this network. What I know is that I can generate data from this particular mathematical model. I can use my MATLAB or C language or any computing program and generate u t, which is a random number between 0 and 1.

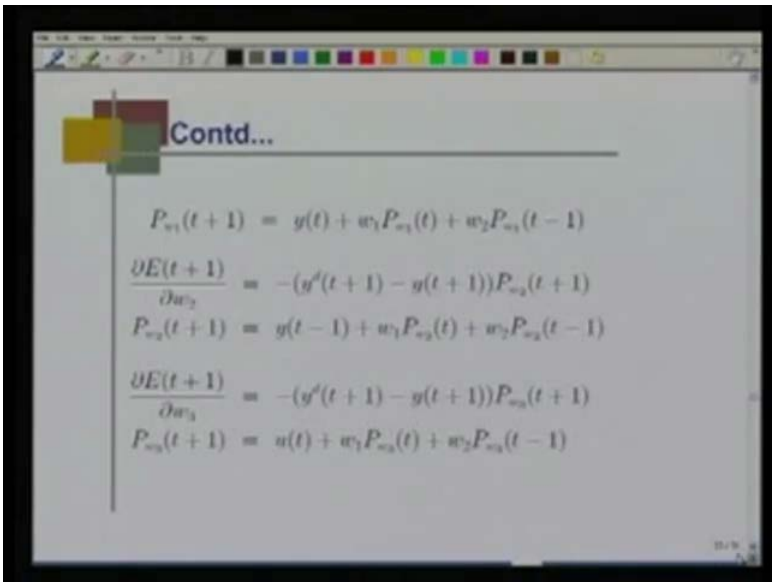
Then, given u_0, u_1, u_2, u_3 and u_4, I compute correspondingly what are y_1, y_2, y_3, y_4, y_5, and so on. I compute this data and give those data online, that is, I compute one, I give one input, find the output, give that here, match the error, back propagate, update the weight, again generate a new data from this model, give it to this network, compute the output, and match the actual output and go on like that. Finally, what I should see is that w_3 should be 0.5, w_1 should be —0.5, and w_2 should be —1. That should be the result.

(Refer Slide Time: 49:06)



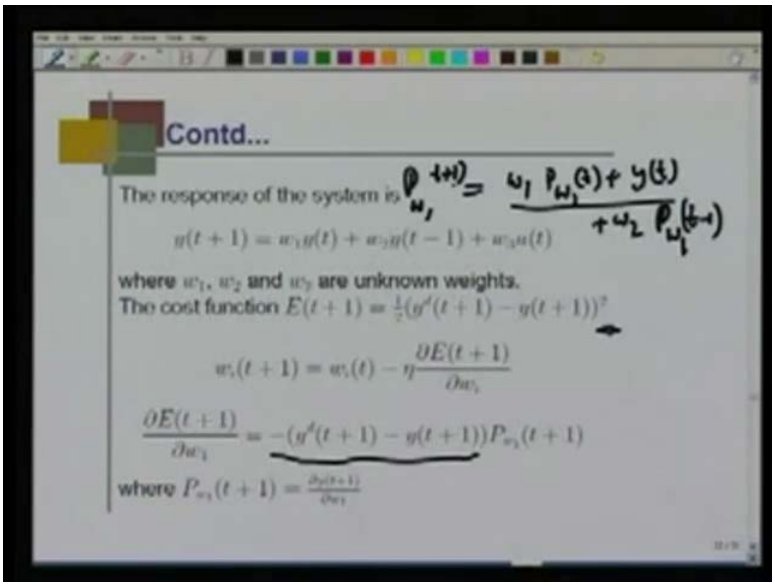
We will see what is happening here. The response of the system is w_1 y t plus w_2 y t minus 1 plus w_3 u t, where w_1 and w_2 are unknown weights. The cost function is instantaneous cost function. So, w_i t plus 1 is w_i t minus this is gradient descent rule (Refer Slide Time: 49:23). As usual, real-time recurrent learning means we put our usual error term into gradient information at t plus 1. We have to now compute P_{w1} t plus 1 in terms of the gradient information in the….

(Refer Slide Time: 49:49)



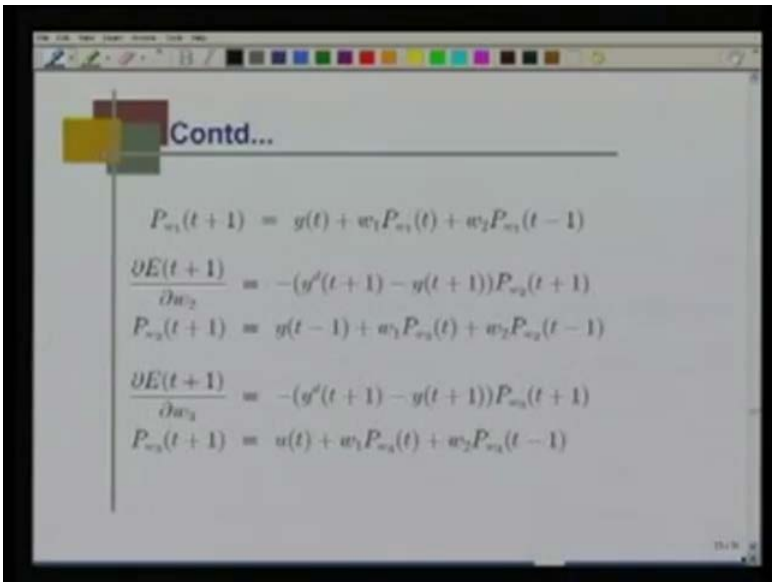
If you look at that, it is very simple here.

(Refer Slide Time: 49:53)



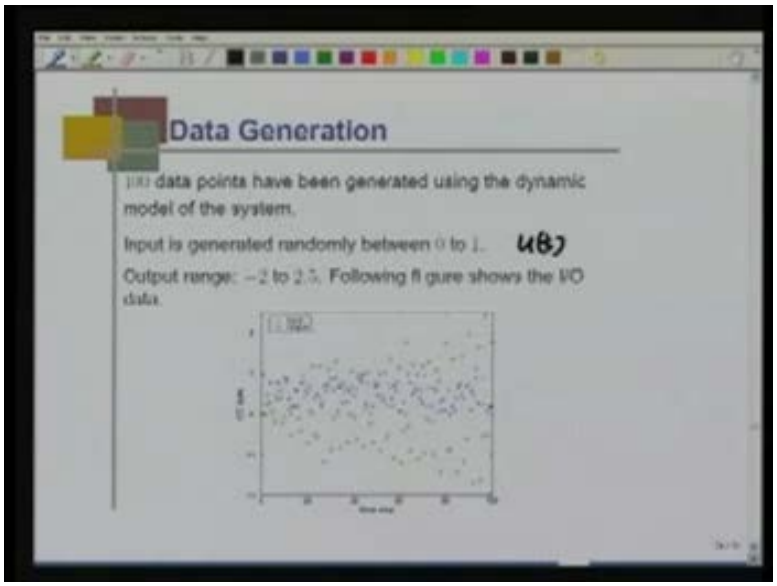
If I want to differentiate this particular thing, del w_y t plus 1 by del w_1, if you see here (Refer Slide Time: 50:02), if I differentiate this with respect to w_1, the first one is…. I am differentiating with respect to w_1, so w_1 into…. If I define, this is simply P_{w1} t plus.… Here also, if I differentiate with w_1, I simply have y t. If I differentiate this with respect to w_1, I get first y t and then w_1 into del y t upon del w_1, which is this, plus w_2 is not a function of w_1, so simply w_2 into P_{w1} t minus 1 plus here, of course, there will be no term because neither of these is a function of w_1. This is actually your this with respect to w_1, that is, P_{w1} t plus 1.

(Refer Slide Time: 51:32)



Now, this is the same thing. y t plus w_1 P_{w1} t plus w_2 P_{w1} t minus 1, which we showed here (Refer Slide Time: 51:40). Similarly, we can find out what is P_w. How many weights do I have? I have three weights. We will have three gradient descent information P_{w1}, P_{w2}, P_{w3}.

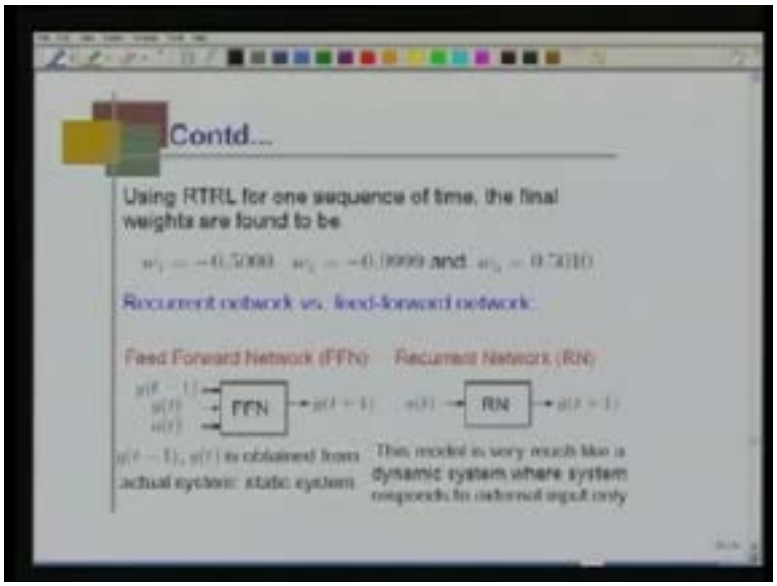
(Refer Slide Time: 51:55)



After we found the RTRL rule, now we will generate the data. What we did is that we generated 100 data points using the dynamic model. Here, u t was a random number between 0 and 1. You

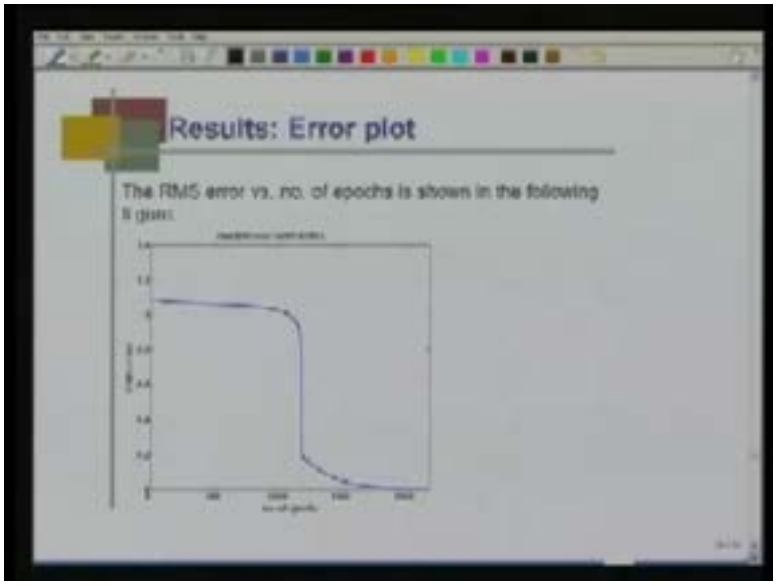
can easily see that all these blues are the inputs and correspondingly, the output was from —2 to 2.5.

(Refer Slide Time: 52:30)



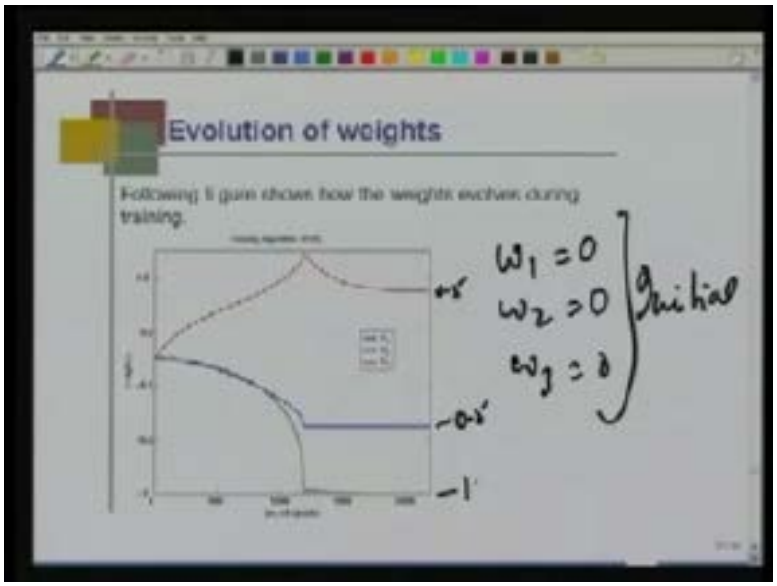
Using RTRL for one sequence of time, these are the final weights. You have 100 data sets. Over that, when you train, finally, you get after training w_1 is —0.5, w_2 is —1 as required, and w_3 is 0.5. What you are seeing here is that in this particular network, I only give recurrent network in real time. My input is only u t, whereas if I would have trained the same data set, I should have three input to the network. So, this is a 3 by 1 and this is a 1 by 1 network. This is more like a dynamic network. So, the recurrent network is a dynamic network.

(Refer Slide Time: 53:25)



This shows the RMS error versus the number of epochs. You can easily see how the RMS error is decreasing and going towards 0 and the number of epochs is almost 1,500.

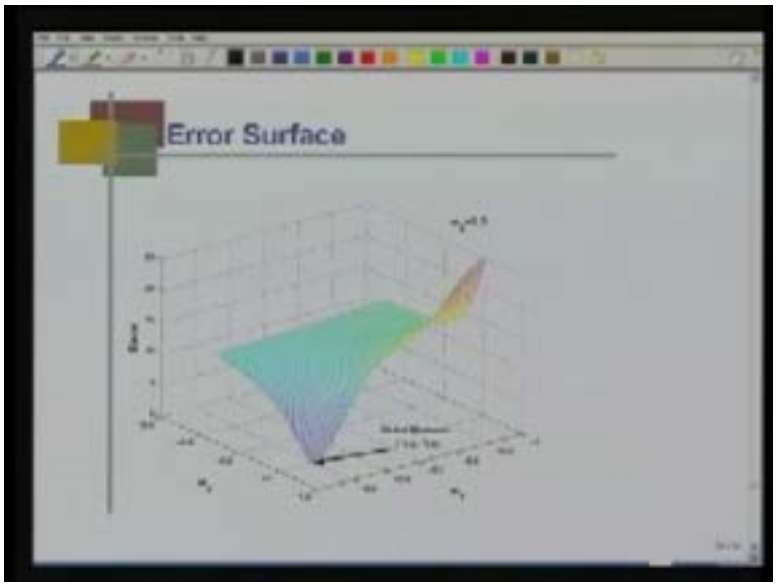
(Refer Slide Time: 53:41)



You see how the weights are evolving. In the beginning, all the weights were initialized to be 0 here. Initially, w_1 was 0, w_2 was 0 and w_3 was 0; this is initial. As the training progressed, the weight got settled at this weight, which is w_3, settled at 0.5; this is 0.5 value (Refer Slide Time:

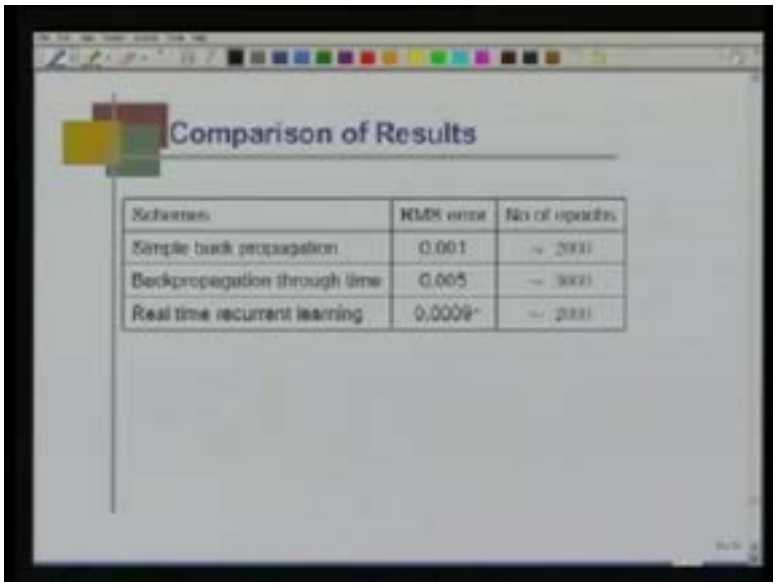
54:12) and w_1 is settled at —0.5, and w_2, which is this one, is settled at —1 – these are the desired.

(Refer Slide Time: 54:29)



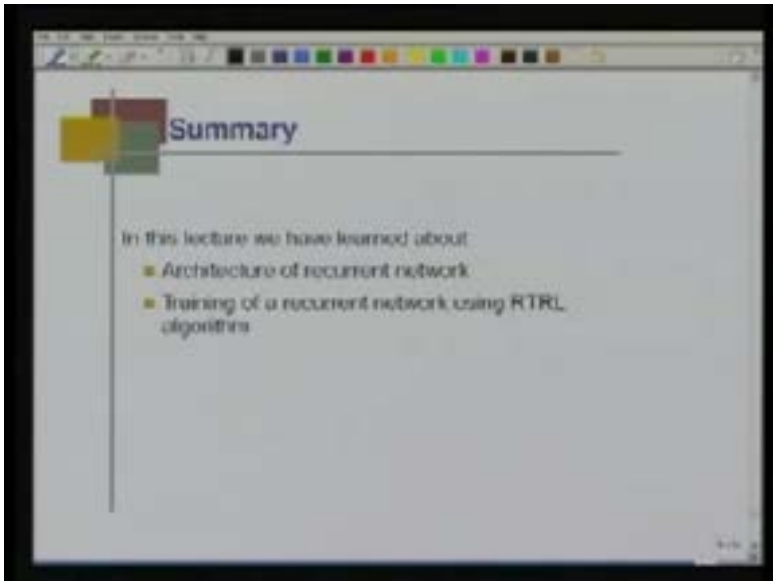
This is the error surface. If you look at the error, how in the error surface, the weights glided to the bottom-most position, the global minimum. In the linear network, we always reach a global minimum using gradient descent.

(Refer Slide Time: 54:44)



We compare the same system that we took. We trained it using simple back propagation through time as well as real-time recurrent learning. After 2,000 epochs, we found a simple back propagation gave 0.001 RMS error, back propagation through time gave 0.005 – even more, whereas real-time recurrent learning is 0.009. You see that the dynamic network does better and it captures the dynamics better.

(Refer Slide Time: 55:34)



In this lecture, we learnt the architecture of a recurrent network. We also presented a training algorithm using RTRL (real-time recurrent learning) and this is an online learning algorithm. We also compared. We took a simple example and on that, we compared how three different learning schemes work. As this is a course on intelligent control, I hope that you appreciate the system. At least in the beginning, I have presented neural network as a system identification tool because that is a better way to look at neural network models from a control prospective. Thank you very much.