**Intelligent Systems and Control**

**Prof. Laxmidhar Behera**

**Department of Electrical Engineering**

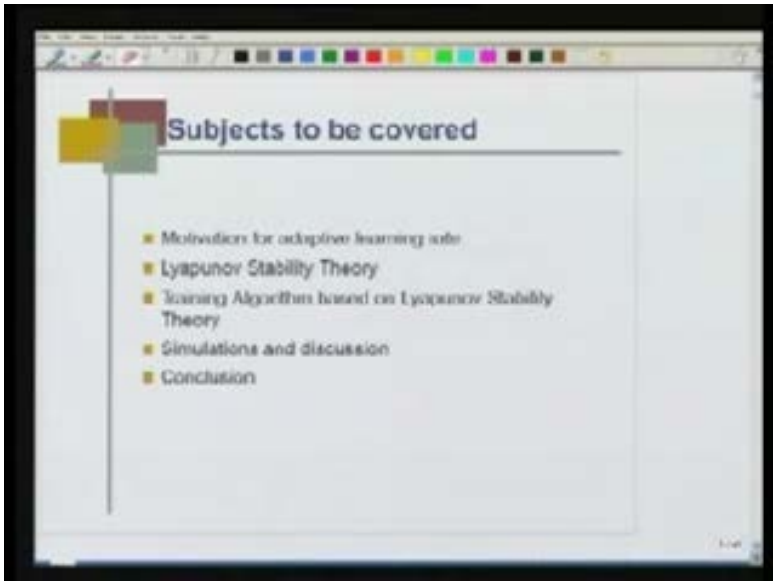
**Indian Institute of Technology, Kanpur**

**Module – 1 Lecture – 7**

**Adaptive Learning Rate**

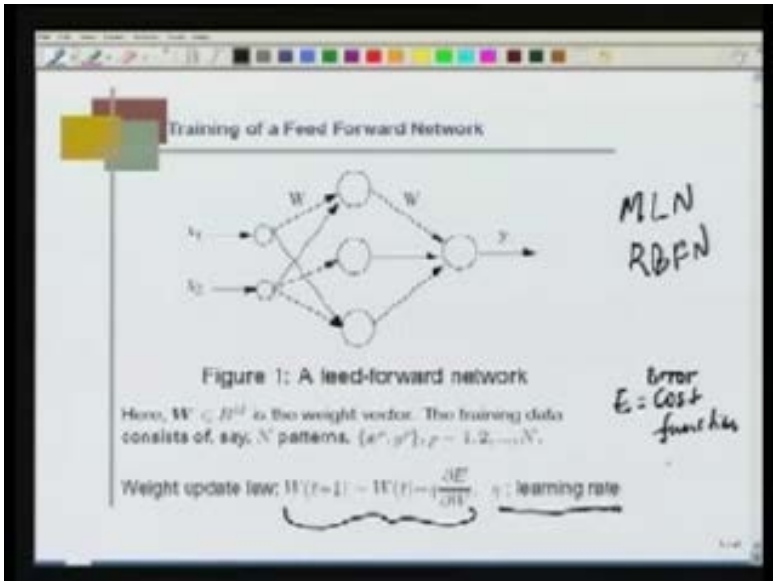
This is the lecture on adaptive learning rate. We have already discussed various types of feed-forward networks and the training that we have used is based on gradient descent technique, where the learning rate eta is a fixed quantity. In this class, we will talk about what is the purpose of making this learning rate adaptive and how we can achieve it.

(Refer Slide Time: 01:05)




The subjects to be covered in this class are motivation for adaptive learning rate, Lyapunov stability theory. We have already discussed about stability theory. This is just to familiarize you in this class so that you can appreciate how to derive this adaptive learning rate using Lyapunov function. Then, some simulations and discussions.

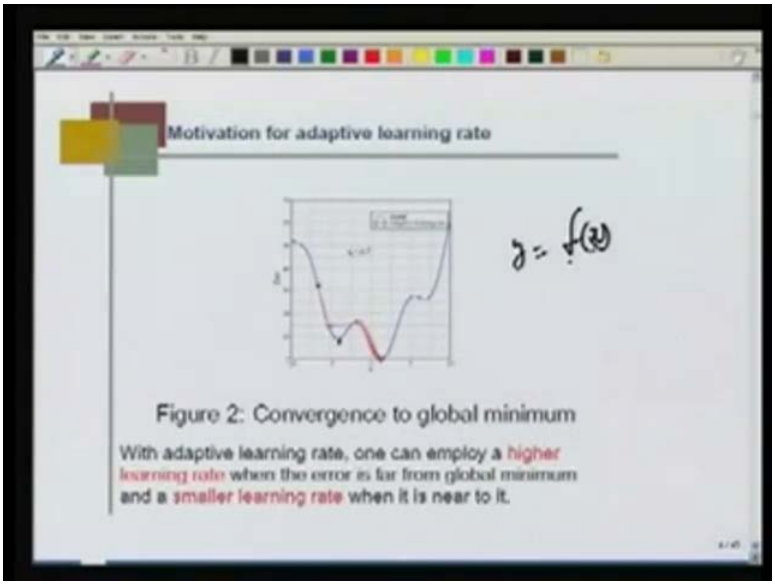
(Refer Slide Time: 01:43)




This is a feed-forward network. This feed-forward network can be a multilayer network or RBFN network. There are various architectures of feed-forward networks. It can be a multilayer network or it can be a radial basis function network. Once you have this kind of structure, the goal is to identify this W, which is the weight vector. What we have done is that we have taken all these weights in the first layer, second layer and as many layers we have in the network, and put them in a vector format, which is W. That means the total number of weights in this network is capital N.

The training data consists of, say, N patterns x p and y p. x p is the vector here, that is, the input vector and y p is the output vector. It is not necessary that we have only a single output or two inputs. This is simply a schematic diagram. The network can have as many inputs as and as many outputs and of course, as many hidden units. The update law for such a multilayer network, radial basis function network has this structure (Refer Slide Time: 03:12), where W t plus 1 is W t minus eta del E upon del W, where eta is the learning rate. This is a gradient descent, del E by del W is the gradient descent, that is, E is the error cost function and that is minimized by this update rule. Let us look at the motivation for adaptive learning rate.

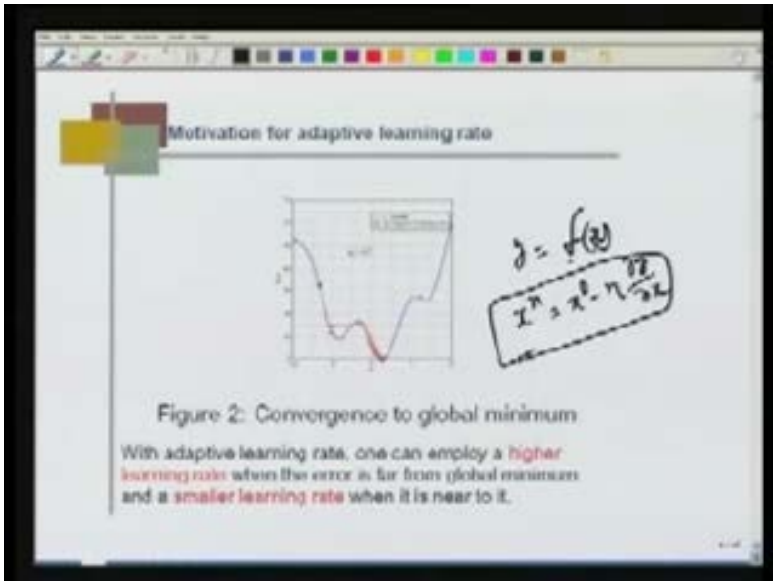
(Refer Slide Time: 03:52)




In general, if I am looking at a function f x…. Let us say y is equal to f x. This is my f x and this is x. When f is nonlinear, then the usual curve can look like this. This is just a representative, this need not be exact. In principle, what this curve says is that this curve has many …. This is one minimum (Refer Slide Time: 04:29), this is another minimum and this is another minimum. In this curve, there are three minimums out of which two are local minimum and this is global minimum, but there can be another function f x for which there can be so many local minimum and many global minimum. So, many local minima and many global minima.

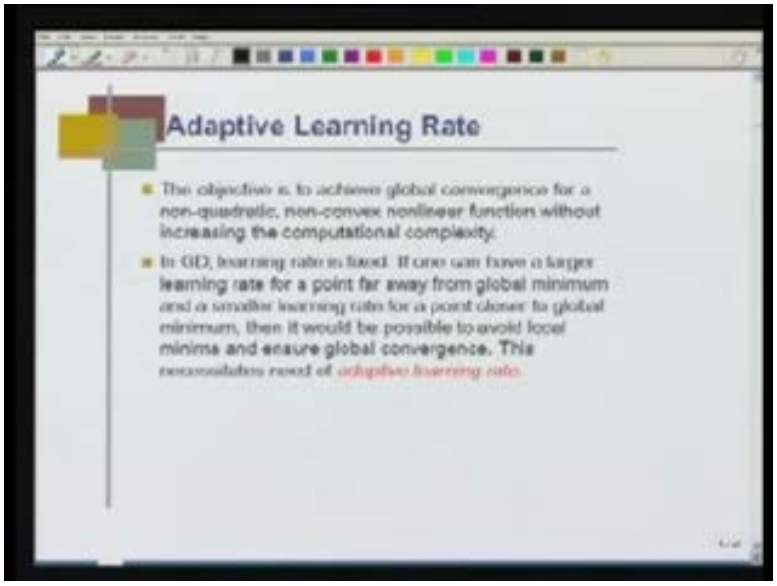
These are the normal characteristics of a nonlinear function f x if I plot with respect to x. The objective is find x such that y is minimum. The objective is how I reach here (Refer slide Time: 05:10), not here, starting from any initial condition. If you see, I use a gradient descent rule and if I start from this point, this red circle and then from here, I glide down and come here at this point (Refer Slide Time: 05:26).

(Refer Slide Time: 05:32)



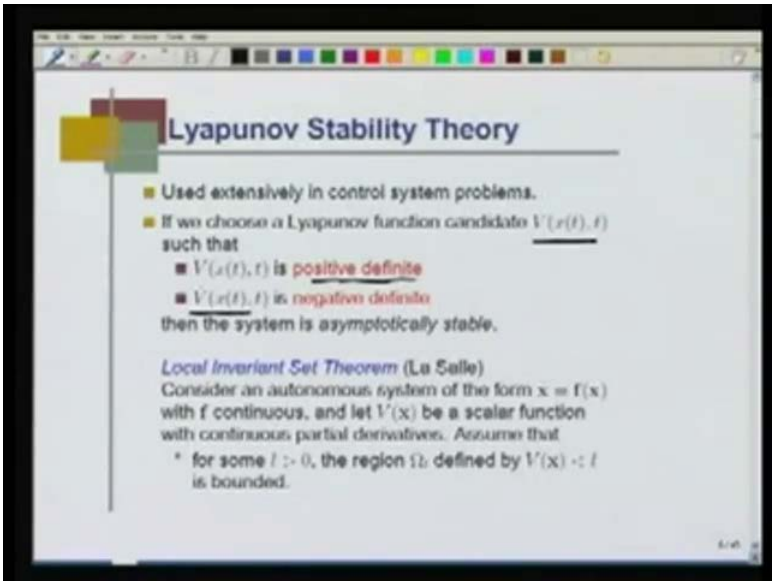

But if I can change this initial position to this point (Refer Slide Time: 05:39), then I can come here by using the rule x new is x old minus eta del y by del x. This is my normal principle of gradient descent that I can come to the local minimum that is nearest to the starting initial point. But such a weight update does not guarantee that I can reach here. Of course, in this case, you can easily see that if I am starting from here, if I increase my learning rate step eta, then this initial position may go to this position (Refer Slide Time: 06:20) by simply changing this eta, a bigger value and then by lowering eta, I can easily reach this. We have discussed these in previous classes. With adaptive learning rate, one can employ a higher learning rate when the error is far from global minimum and a smaller learning rate when it is near it, but how do you do it? There are certain heuristics methods that people have tried.

(Refer Slide Time: 06:47)



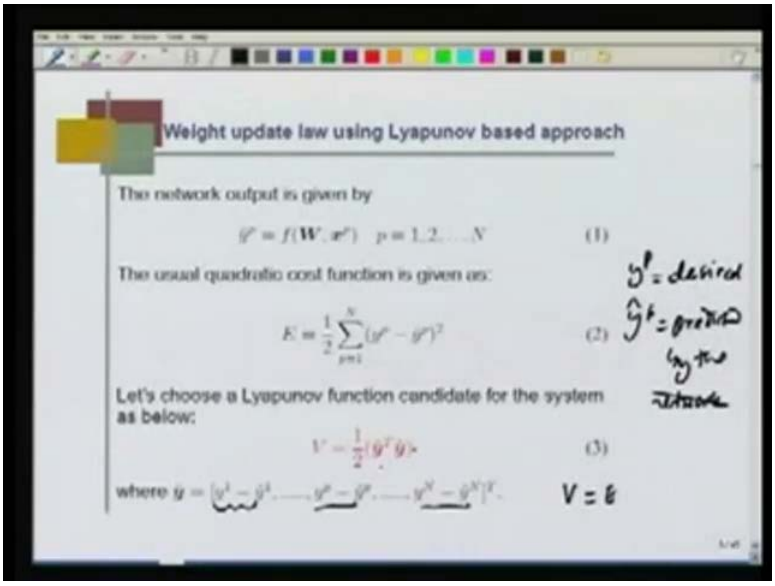

But today, we would like to give a comprehensive approach on how to compute this adaptive eta such that even if I start from the initial weight (Refer Slide Time: 07:00), which is nearer to a local minimum, I can still reach the global minimum – that is the objective. The objective is to achieve global convergence for a non-quadratic, non-convex, nonlinear function without increasing the computational complexity. In gradient descent, the learning rate is fixed. If one can have a larger learning rate for a point far away from a global minimum and a smaller learning rate for a point closer to the global minimum, then it would be possible to avoid local minima and ensure global convergence. This necessitates the need for adaptive learning rate. Today, in this class, we will derive how to comprehensively compute eta, the adaptive learning rate such that we reach…. Of course, our algorithm does not guarantee that we will have a global convergence, but it is better than gradient descent. We will be using a Lyapunov function. What is a Lyapunov function?

(Refer Slide Time: 08:12)




For a system that is defined in the state space x t, V x is a Lyapunov function if it is positive definite and it grows, that is, V x t increases as x t increases. Once we have selected such a Lyapunov function and if I compute the rate derivative of that Lyapunov function V dot x, if it is negative definite, then the system is asymptotically stable. This is the Lyapunov function. Another condition is that if V dot becomes 0, then x t also converges to 0.

(Refer Slide Time: 09:01)

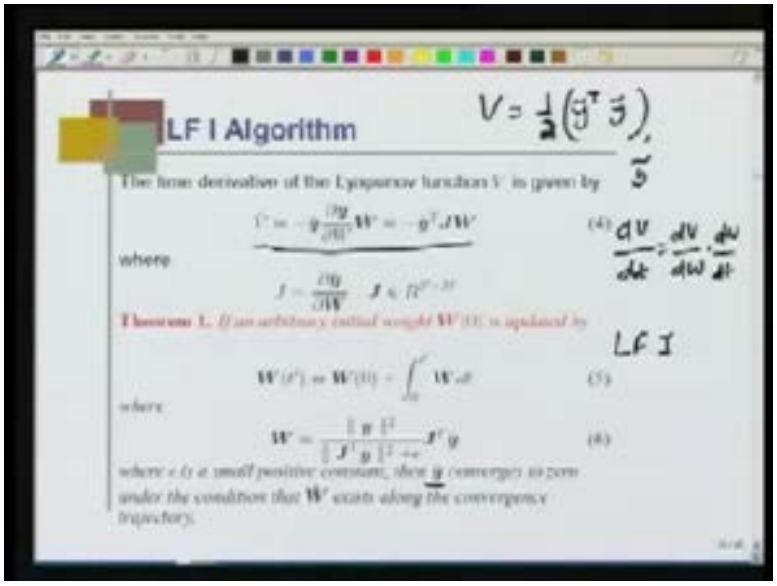


Weight update law using Lyapunov based approach. Normally, given a network that is characterized by W…. We have already discussed all these things. The network is characterized by the weight vector W and the input vector is x p, then the network output is y p hat. This is network prediction and p stands for pattern. I have N such patterns. The usual quadratic function E given for this kind of network for training the weight vector W is y p, which is desired, and y hat p is the actual one.

y p is desired and y hat p is predicted by the network. You can see that this is the usual quadratic function that we have already discussed. This is for N patterns and if I have also multiple outputs, then I can put another index here and make a sum there. For such a network, let us choose a Lyapunov function candidate for the system as below. V is half y tilde transpose. It implies that here, I have this y tilde is actually the error for the pattern 1; y tilde is a vector of error function – the error that occurs when a network predicts given some input x p with specific W. This is the error for the pattern 1 (Refer Slide Time: 11:03), this is the error for pattern p and error for pattern N.

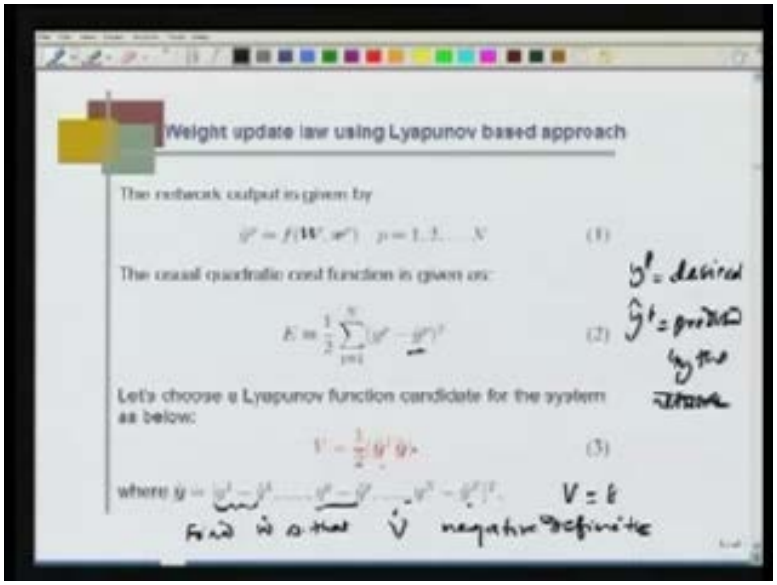
We have capital N sets of patterns and for each pattern, we compute the error and each error is an element of y vector y tilde. Obviously, V is…. You can easily see that whatever I am writing here and this E are the same. In this case, V is the same as E – we have just written in a vector format. This expression has been written in a vector format in V and we say let this be a Lyapunov function. Then I try to differentiate V dot.

(Refer Slide Time: 11:54)




You check here.

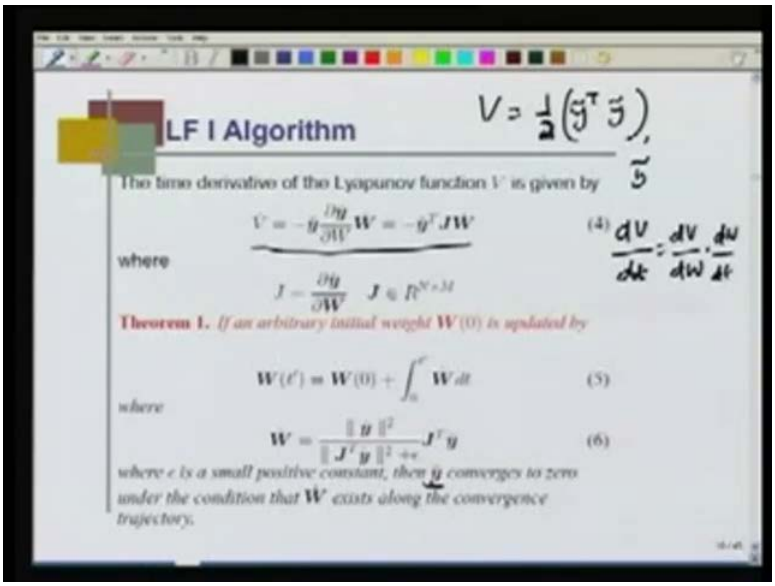
(Refer Slide Time: 11:57)




This is my Lyapunov function and in this, there is a term called y hat p (Refer Slide Time: 12:03) and this y hat p is at W. The objective is how can I find out W dot? The objective is to find W dot such that V dot is negative definite – this is the objective. Find W dot such that if I differentiate V, V dot is negative definite. That is how we deal with a nonlinear function. You

can easily see this is a nonlinear dynamics, because V is a nonlinear function and y hat p is a function of W, nonlinear function and I am now saying W dot, that is, the evaluation of W follows certain dynamic law. Then, this becomes a (Refer Slide Time: 13:18) dynamic equation and if V dot can be negative definite, then I can say that the W dot that is given to me is the weight update law that (Refer Slide Time: 13:30) stability.

(Refer Slide Time: 13:36)



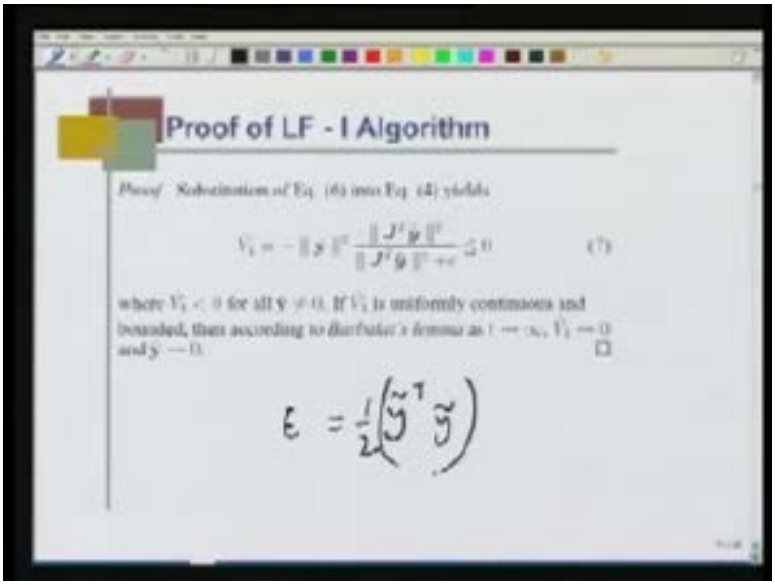

The time derivative of the Lyapunov function V is given now – V dot is…. You see that V was half y tilde transpose y. This is a scalar function and y is a vector, where y tilde is the vector of all the errors for each pattern. V dot is minus y tilde del y hat by del W dot. You can easily see here that if I differentiate this V dot, y tilde and then differentiation of this quantity with respect to W and then, W dot. I can write this equation as minus y tilde transpose J W dot. You can easily see that this…. I hope this expression is clear to you (Refer Slide Time: 14:57), where I simply differentiate this V. The negative sign comes because y tilde is y actual minus y hat. When I differentiate that, I get minus del y hat by del W and W dot.

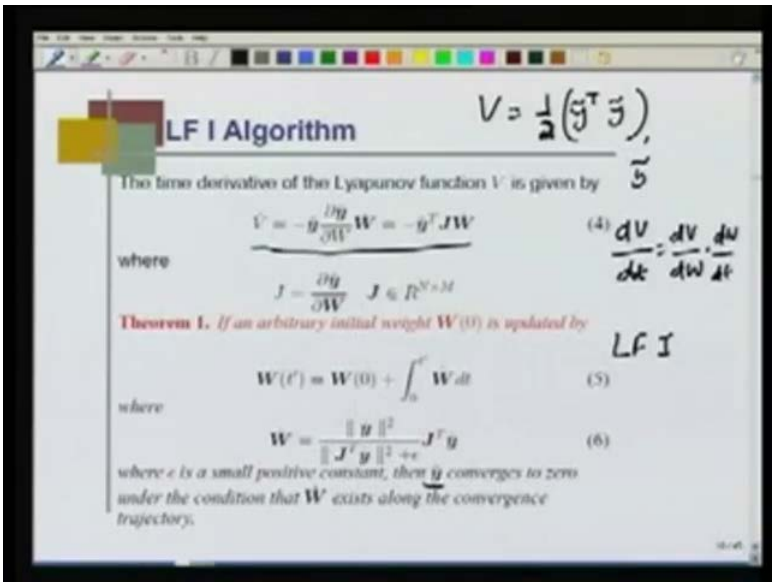
The relation that I have added is dV by dt is dV by dW into dW by dt. What is this? W dot is dW by dt and this particular expression, this part, you can easily see that this is a vector (Refer Slide Time: 15:50) and this is another vector, but this is a row vector. Of course, we can always say that y is a vector, W is another vector, so obviously this is actually a matrix, which is this. I

define this quantity as J and J is equal to del y upon del W – this quantity. The dimension of this matrix is N into M because W has dimension capital M and y has dimension N. I hope now you understood what V dot is. V dot is this quantity (Refer Slide Time: 16:52).

Now, the theorem is if W is updated by this equation, the W t dash is W 0 plus 0 to t dash W dot dt. At every time t, I compute this relationship, where this W dot is given by this expression, which you are probably not so familiar with, but y tilde is norm square. Norm square means y tilde is a vector. So you take each element, square it, add them and take the root of that – that is the norm. If you do not take a root, norm square, so W dot is this norm square. J transpose y tilde is another vector, again norm square, plus this epsilon is a small positive constant and then J transpose y tilde. If this is the case, this is W dot, then, y tilde converges to 0 under the condition that W dot exists along the convergence trajectory. That is what we need. We need that this y tilde, the vector y tilde should converge to a zero vector under the condition that W dot exists along the convergence trajectory. This is the theorem. Once we have a theorem, we must prove that theorem. How do you prove it?

(Refer Slide Time: 18:34)

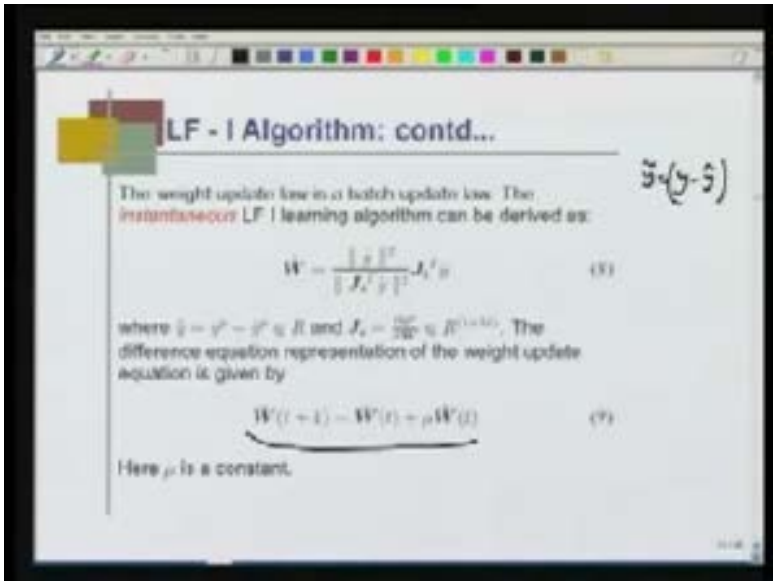





We will talk of this theorem as Lyapunov function I (Refer Slide Time: 18:41), because we prove it using the Lyapunov theory, Lyapunov function concept. What we do is we take this equation 6 and you replace this in 4 and compute V dot. If we do that, we get V dot to be y tilde square and this quantity (Refer Slide Time: 19:09). You can easily see that this quantity is always either positive…. you take the negative the other quantity. This is a positive quantity, this is another positive quantity and this is also another positive quantity. Hence, this quantity is always either negative, because negative sign is there or 0. That assures us if $V_1$ dot goes to 0, then this quantity also will go to 0.

If $V_1$ dot is uniformly continuous and bounded, then as t tends to infinity, $V_1$ dot tends to 0. If $V_1$ dot tends to 0, you can easily see that y tilde will go to 0. That is the proof. The objective was that our error cost function, which is simply half y tilde transpose y tilde…. If y tilde goes to 0, then the cost function becomes 0, that is, the function is minimized.

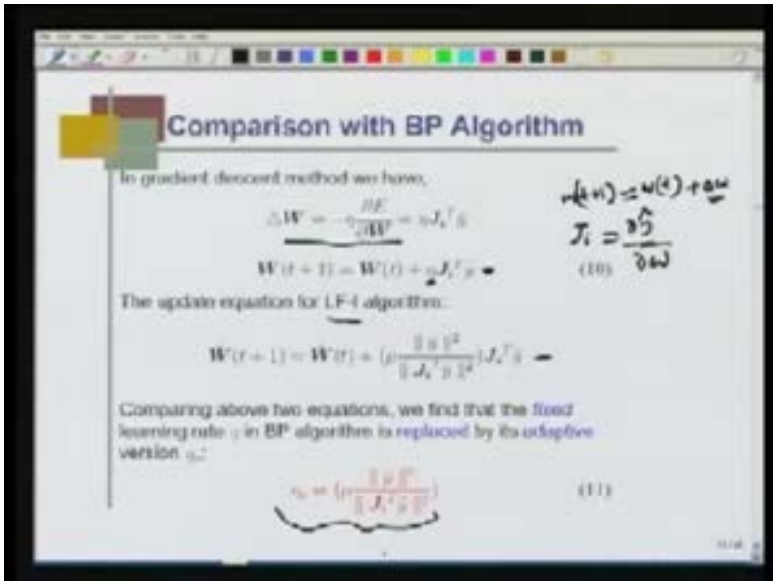
(Refer Slide Time: 20:37)




The weight update law is a batch update law that we discussed now. The instantaneous LF-I learning algorithm can be derived as…. For instantaneous, what we do is we do not take the vector y tilde, we simply take a scalar. y tilde is simply y minus y p. This is no more a vector. This is for instantaneous quantity. W dot is this quantity. If I consider what should be the weight update law for instantaneous update…. We have already discussed earlier that there is batch update, there is instantaneous update. For batch update, we consider all the patterns and for instantaneous update, we consider the recent pattern, current pattern. This becomes the update law using the same Lyapunov function approach.

If I look at this W dot, how will I update this? If I take a discrete equivalent of this equation, then this is the discrete equivalent (Refer Slide Time: 21:36) – the difference equation form of the continuous update law, which is W hat t plus 1 is W hat t plus mu W dot t. This is how the weight update is done, where mu is a suitable small constant such that this equation approximates equation 8.

(Refer Slide Time: 21:59)



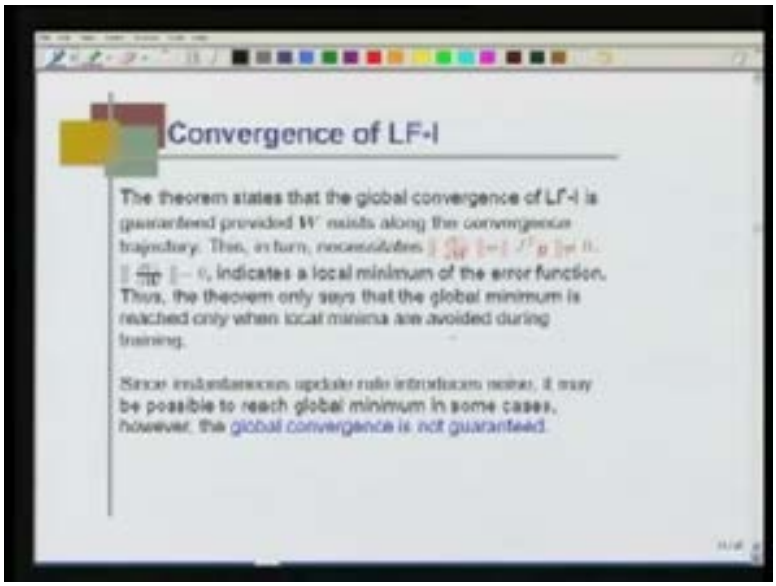

We derived the weight update law W dot. That would ensure that y tilde converges to 0 if W dot exists along the convergence trajectory. That was our theorem and we proved it. Let us go in depth of this algorithm. If you look at the BP algorithm, back propagation algorithm, delta W is minus eta del E upon del W. What is delta W? I always write W t plus 1 is W t plus delta W. You can see that this is the gradient descent law, gradient descent method (Refer Slide Time: 22:57) and this gradient descent method can be written as eta $J_i$ transpose y tilde where this $J_i$ is the same as the $J_i$ is del $y_i$ by del W.

You can easily that this is $J_i$ (Refer Slide Time: 23:23) $J_i$ is del y by del W. This is a row vector because y is a scalar. There is no i there (Refer Slide Time: 23:37), so del y hat by del W is here. If I write the equation, this equation is the same as this equation. This eta $J_i$ transpose y tilde is the same as minus eta del E upon del W. Then, I write this equation and using the recent algorithm which I call LF-I, we got an equation which is $W_t$ plus 1 is $W_t$ plus mu y tilde norm square by $J_i$ transpose y tilde norm square into $J_i$ transpose y tilde.

You can easily see that this back propagation algorithm was very popular and the algorithm we have derived is remarkable (Refer Slide Time: 24:33) same identity, except that eta is replaced. In back propagation, this eta is a fixed quantity and in this Lyapunov function based weight update law that we just now derived is this quantity, whereas this quantity is a time-dependent

quantity. That gives us a notion that using the Lyapunov function, we can actually derive an adaptive learning rate, which is eta. Comparing the above two equations, we find that the fixed learning rate eta in BP algorithm is replaced by its adaptive version, which is eta adaptive. This is the first time (actually, this is our own work) we derived an adaptive learning rate using…. Comprehensively, we found this an expression for adaptive learning rate using Lyapunov function method.

(Refer Slide Time: 25:53)

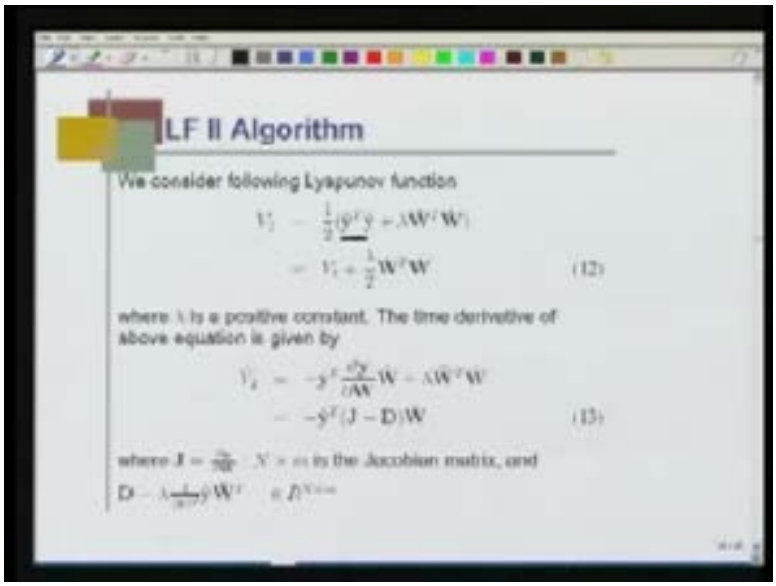



The convergence of LF-I is that we said that… we showed that y tilde converges to 0, but there was a condition that if W dot exists along the convergence trajectory, then our theorem is right. But W dot may not exist along the convergence trajectory. Then, our thing does not hold good. The theorem states that the global convergence of LF-I is guaranteed, provided W dot exists along the convergence trajectory. This, in turn, necessitates that del $V_1$ by del W, the norm of that, is J transpose y tilde is not equal to 0.

This says that the W dot exists along the convergence trajectory (Refer Slide Time: 26:51) and for this existing convergence trajectory meaning that this should always exist – J transpose y tilde should always exist. This implies that J transpose y tilde should not be 0 or del $V_1$ upon del W is 0, which indicates a local minimum of the error function. Thus, the theorem only says that the global minimum is reached only when local minima are avoided during the training. Since

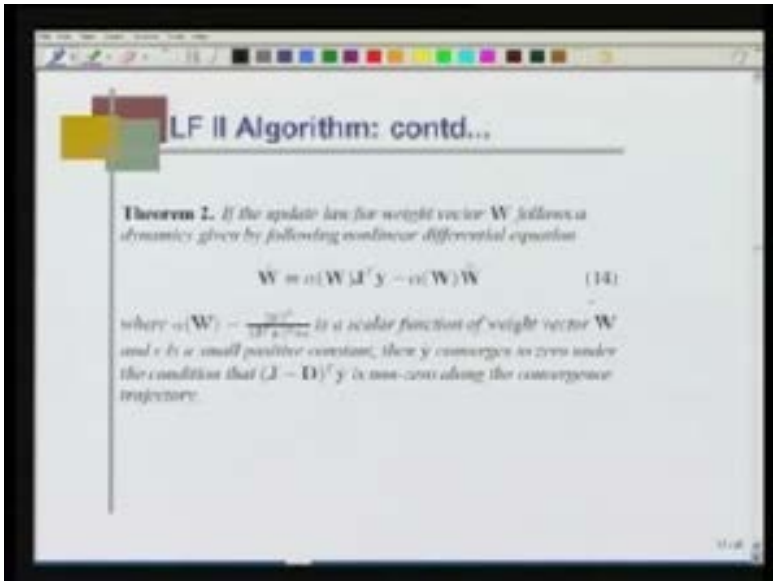
instantaneous update rule introduces noise, it may be possible to reach global minimum in some cases. However, global convergence is not guaranteed. This is the implication of the theorem that W dot exists along the convergence trajectory.

(Refer Slide Time: 28:43)



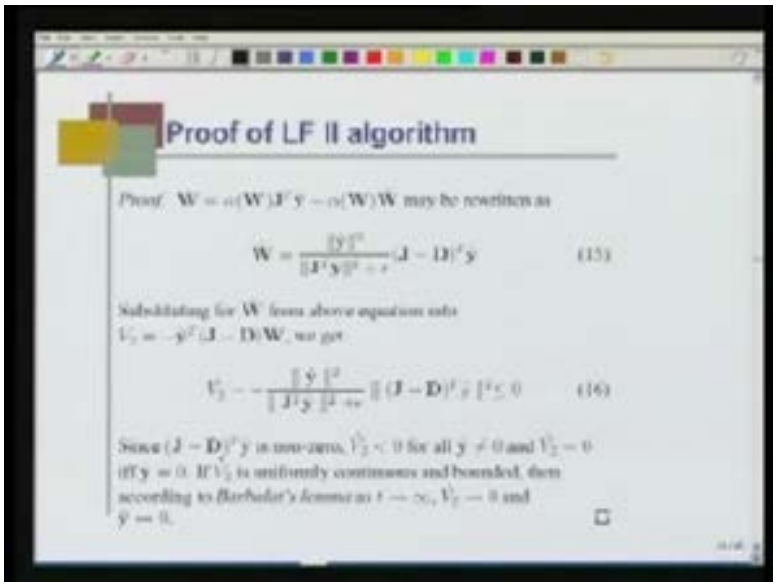

This is a second version of that same algorithm where our Lyapunov function $V_2$ is the earlier Lyapunov function. This was the earlier Lyapunov function we took and we added a term, which is lambda into W dot transpose W dot. This term was added into the error function $V_2$. Again, if we follow the same method, we find out $V_2$ dot, then this can written in this form, which is minus y tilde transpose into (J minus D) W dot, where J is del y upon del W and D is this quantity, that is, lambda into 1 upon y tilde norm square y tilde W double dot transpose. If I put this, then this $V_2$ dot, which is computed here…. This is $V_2$, this is $V_2$ dot and this is the same quantity as this.

(Refer Slide Time: 29:00)



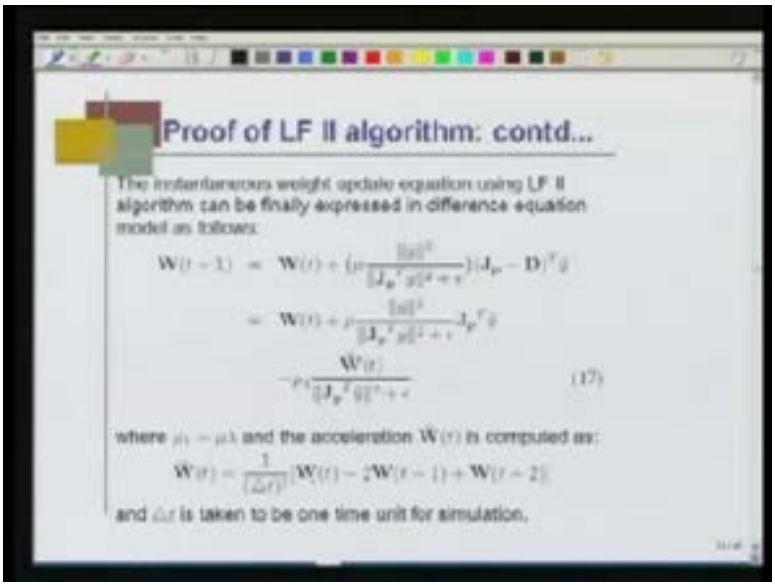

Now, we propose another theorem. If the update law for weight vector W follows a dynamics given by the following nonlinear differential equation, which is 14, where this alpha W has this form y tilde norm square upon J transpose y tilde norm square plus epsilon is a scalar function of weight vector W and epsilon is a small positive constant, then y tilde converges to 0 under the condition that J minus D transpose y tilde is non-zero along the convergence trajectory. This should be non-zero along the convergence trajectory.

(Refer Slide Time: 30:24)



We take this W dot and this W dot can be written in this form now. From here, I can write this equation in this form. Substituting for W dot from this equation, if I replace this W dot here, which is $V_2$ dot, then $V_2$ dot becomes this quantity. You can easily see that this is a positive quantity or 0. This is again a positive quantity and this is another positive quantity or 0 (Refer Slide Time: 30:19). In that sense, since (J minus D) transpose y tilde is non-zero, $V_2$ dot is less than 0 for all y tilde less than or equal to 0 and $V_2$ dot is 0 if y tilde equal to 0. If $V_2$ dot is uniformly continuous and bounded, then according to Barbalat's lemma as t tends to infinity, $V_2$ dot tends to 0, implying y tilde will converge to 0, which was the theorem. The theorem is proved that y tilde you will again converge to 0, provided the condition….

(Refer Slide Time: 31:02)

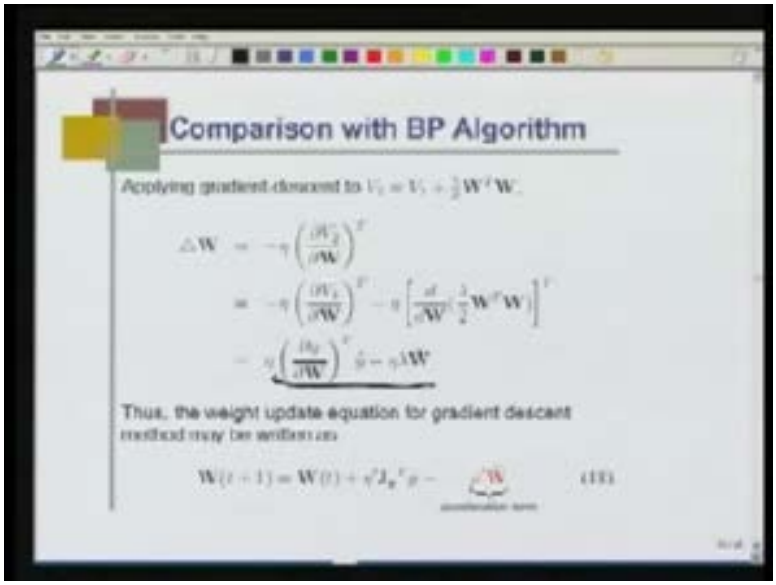



The condition is (J minus D) transpose y tilde is non-zero along the convergence trajectory. This is necessary because if this is not 0 (Refer Slide Time: 31:15), if this becomes 0, then W dot does not get updated. We say that W dot should exist (Refer Slide Time: 31:21). For W dot existing here, this quantity should not become 0 (Refer Slide Time: 31:32). If this becomes 0, then obviously W dot becomes 0 and then weights cannot move along. The trajectory of weight update becomes stagnant and you cannot have further. So, this should not become 0; if this becomes 0, then this theorem does not hold good. This can be 0 in local minima, that is, for LF-I; for LF-II, we showed now the same thing (Refer Slide Time: 31:56) that this quantity also should not become 0. As long as this is not 0, y tilde will converge to 0.

We derived two Lyapunov function based algorithms, but in both the theorems, we showed that y tilde will converge to 0, but there is a condition. In the first one, the condition was that $J_i$ transpose y tilde should not become 0. Here, (J minus D) transpose y tilde should not become 0 and because of these conditions, global convergence is not achieved. Anyway, the weight update law that we talked about, the difference equation version of that weight update law is this particular format, which is equation 17.

This can be written in terms of two forms. W t plus 1 is W t and there is some quantity here, which is a time-varying quantity, then J transpose P y tilde. This is the instantaneous and that is why P has come – P for each pattern. Minus… again, this is another quantity, where $mu_1$ is mu plus lambda and that quantity is written – again, another time-varying quantity into W double dot. This is a gradient term (Refer Slide Time: 33:42) that has some input and this is W double dot into some constant, where W double dot is computed using this formula, which you can easily derive using Euler's method.

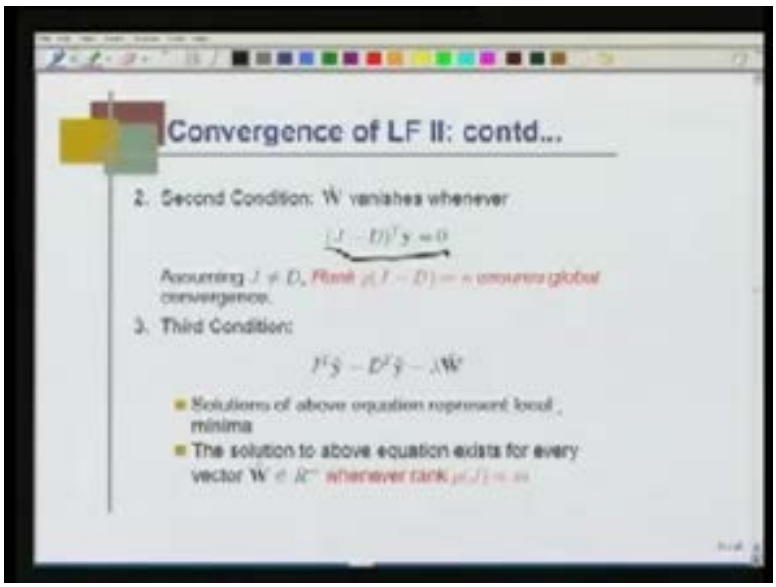
(Refer Slide Time: 34:02)




If I use the same cost function and apply back propagation, I get this equation. The recursive term is this, equation 18. Now also, we are getting the same thing, but we are now getting eta dash and mu dash are time-varying quantities, that is, the adaptive quantity using Lyapunov function. The Lyapunov function or the method that we have described gives us weight update

law, which is similar to gradient descent, but…. that but is very important because the learning rate associated is the adaptive map. Normally, in back propagation, we call this term the acceleration term, but our mu dash is an adaptive acceleration term and eta dash in back propagation is a fixed learning rate, whereas in our case, it is an adaptive learning rate.

(Refer Slide Time: 35:12)

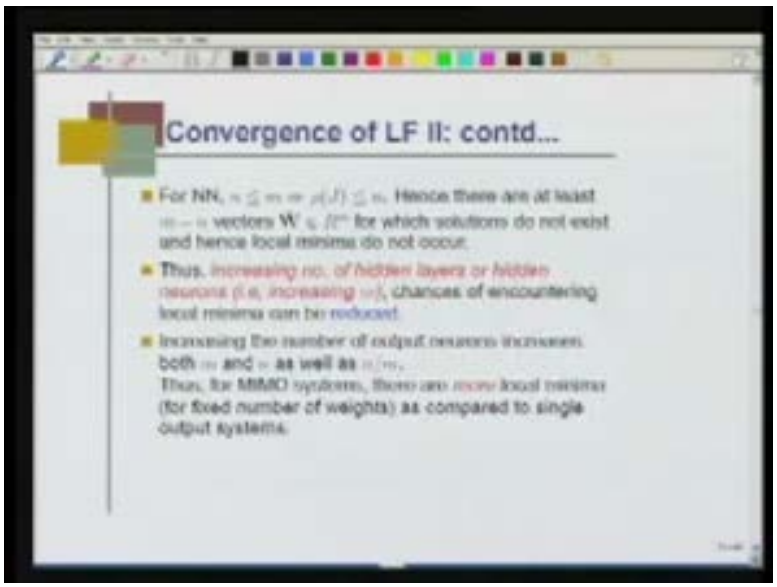



I will not go into details of this because this requires a little bit of more introspection and you really need to understand and go into detail about it. Simple classroom teaching will not help you to understand this. In this particular class, I am just telling you or I am giving you or I am delivering this lecture to just let you know or to inspire you to think about the methodology that will help you to get a better convergence algorithm. The convergence of LF-II is again the same thing that this should not become 0 (Refer Slide Time: 36:02) and if this does not become 0, then it ensures global convergence, but we cannot guarantee that these cannot become 0. This can become 0.

You can easily see that J transpose y tilde is D transpose y tilde. If this is the case, then the solution of the above equation represent local minima. When this is 0, this is the local minima it reaches. We said that this should exist. That means this should not become 0 for global convergence (Refer Slide Time: 36:44), but W dot vanishes whenever this is 0. If we assume J is not equal to D, then rank of J minus D equal to n ensures global convergence because if J is not

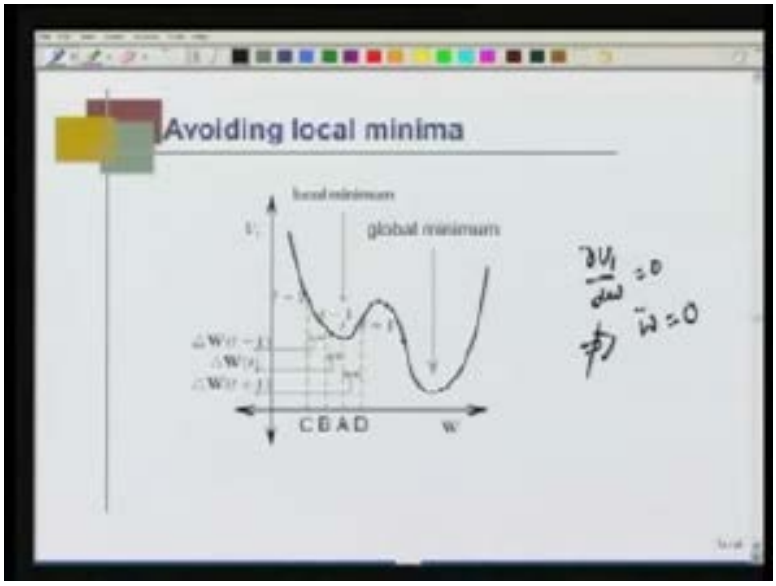
equal to D, it implies this is 0 only when y tilde is 0 and that is global convergence. At global convergence, y tilde is 0. The other condition is that when J transpose y tilde…, but this case of course is for… when this is true, the solution of the above equation represent local minima and then, the convergence is not global.

(Refer Slide Time: 37:42)



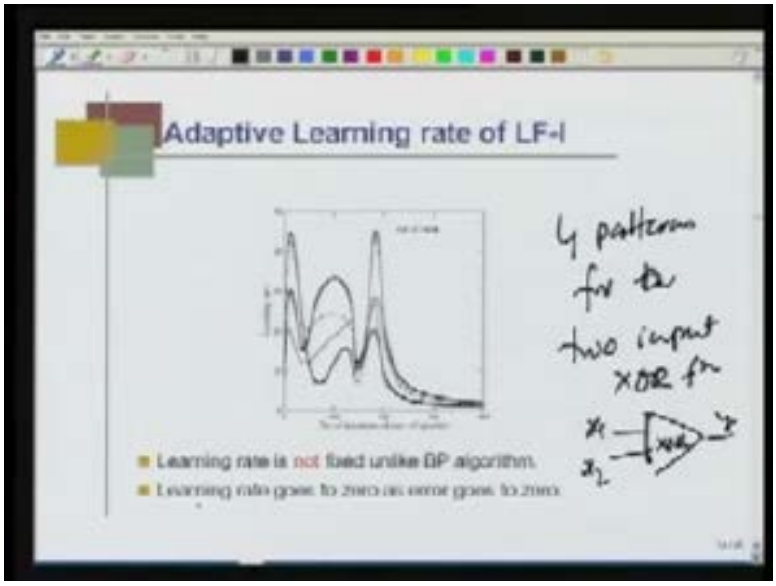

For a neural network, n is much much less than m. There are at least m minus n vectors for which solutions do not exist. Hence, local minima do not occur. Increasing the numbers of hidden layers or hidden neurons, chances of encountering local minima can be reduced because of this particular statement. Increasing the number of output neurons increases m and n, as well as n by m. Thus, for MIMO systems, there are more local minima as compared to single output system. So more complexity, more local minima.

(Refer Slide Time: 38:27)




This is one of the interesting parts. I will not go in detail here but all that I will tell you is that using Lyapunov function, the second Lyapunov function, LF-II algorithm (the first one is LF-I, the second one is LF-II), one can show that because of the acceleration term that we are adding, most of the time… because whenever this quantity del $V_1$ upon del W is 0, that does not ensure that W double dot… Whenever del $V_1$ by del W is 0, it does not imply that W double dot is 0. This fact actually ensures that this term can actually drive away this local minimum. It can bring the weight from this local minima and it can come here. We can avoid local minima, not always but in some cases using LF-II algorithm by adding that extra term, which is half W dot transpose into W. That extra term helps us because that introduces an acceleration term in our learning algorithm. That acceleration term can take away the weight from being stuck in the local minima towards the global minima.

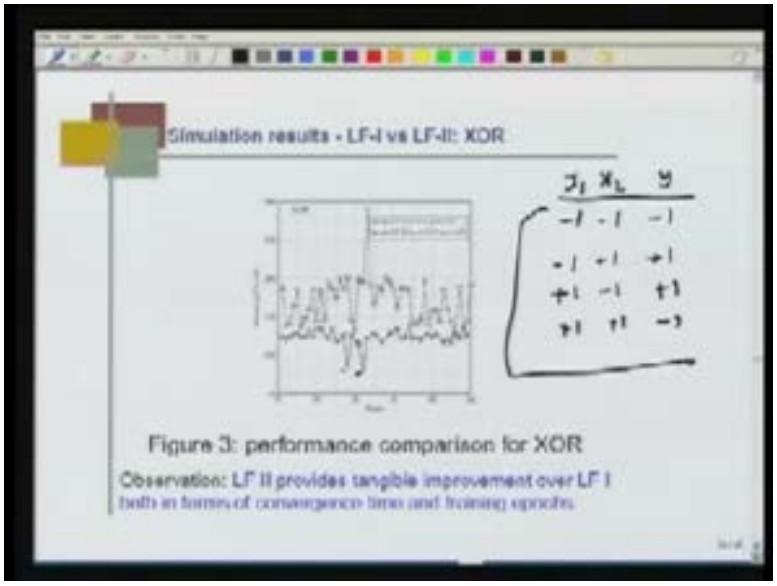
(Refer Slide Time: 40:08)




We talked about adaptive learning rate. What is this adaptive learning rate? Let us consider the LF-I algorithm and in this algorithm, if we apply this algorithm to an XOR function…. All of you know what an XOR function is – when both input are equal, then the output is 0 and unequal, the output is 1. That is the XOR function. We have four patterns in the XOR function for two-input XOR function; four patterns for a two-input XOR function, that is, I have this. This is XOR, $x_1$, $x_2$ and y. If I train this function, if I train a network to learn this function, then when I plot this eta function….

You see for each pattern, I have four curves here – one, two, three and four (Refer Slide Time: 41:32) and you can easily see this eta is increasing and decreasing until it goes to 0. What you are seeing are two important things: learning rate is not fixed, which in BP is a fixed quantity, and learning rate goes to 0 as error goes to 0 – this is important. That is what is important – how the learning should be done. Learning should be done… in the beginning, eta should be very high or whatever is necessary, but as the learning is over for the patterns and the network has already learned, then eta should be 0. Why should eta be there? But for back propagation, the eta is always fixed. This is important. What is the meaning of adaptive learning rate? It has two important properties: eta is time varying and it has a property that y tilde converges to 0, eta also becomes 0.

(Refer Slide Time: 42:39)




We will now talk about simulation results. This is your LF-I versus LF-II, the XOR function. Let me first summarize. We proposed two theorems for weight updates in a feed-forward network – it can be a multilayer network or it can be a radial basis function network. We said that these weight update algorithms will ensure that y tilde converges to 0, that is, global convergence provided two conditions …. There is a condition for each theorem that has to be satisfied. In one case, while the weights are sliding along the convergence trajectory, they should not be stuck in the local minimum. This is the first theorem. The second theorem says J minus D transpose y should also not be 0. Again, that solution also ensures us to be in local minima if it becomes 0. Now, the problem is which is better – LF-I or LF-II? We have some idea that LF-II can avoid local minimum. Let us do simulation and see whether the theory we predicted is true or wrong.

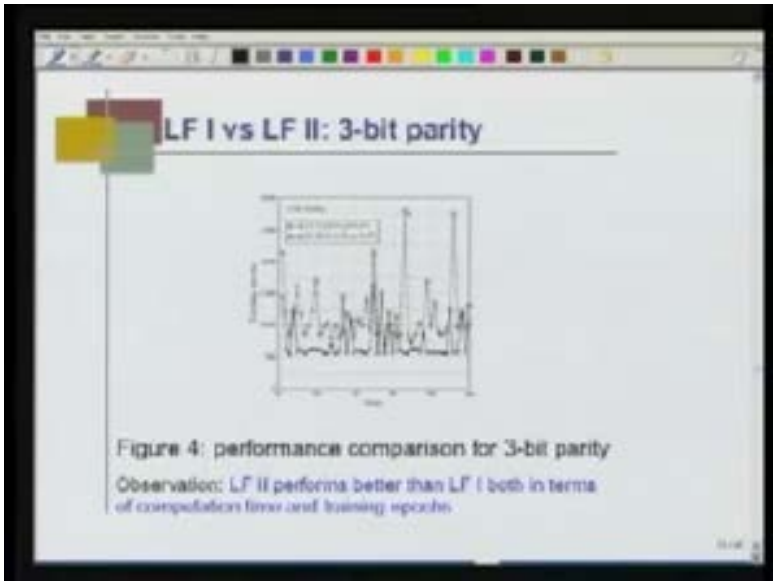
See what we have done here for XOR function. (Refer Slide Time: 44:23) feed-forward neural network, that is, a multilayer neural network and using that neural network, what we did is we started from this different initial condition W and for every run, we wanted to see how many training epochs were necessary for each algorithm for convergence. If you look at the top one, which is the broken line, that represents the number of training epochs. One epoch is training about four patterns. In the XOR function, I have —1, —1 is —1; —1, +1 is +1. This is my $x_1$, this is my $x_2$, this is my y. Then, +1, —1 is +1 and +1, + 1 is —1. This is my XOR function and if I train this XOR function in a multilayer network using my Lyapunov function-I algorithm,

then from every different initial condition, because initial weights are all random, for each initial condition I get.… You see that these are the number of patterns here and one epoch is these four patterns.

I give these four patterns to my network, train it and again, I repeat it. Like that, if I train almost… here, it is almost near 200 and here, it is around 120 or so, it is varying; sometimes, it may also go to up to 300. You can easily see the number of training epochs necessary to train an XOR function using LF-I is always a varying quantity and there is a good deal of fluctuation – that is almost as 120 training epochs to 300 training epochs, whereas if you look at LF-II, it is almost constant. This is a straight line except these fluctuations here – this is one fluctuation, this is another fluctuation (Refer Slide Time: 47:09), but otherwise, they are very fixed quantities. This is always below the training epochs that are necessary using LF-I.

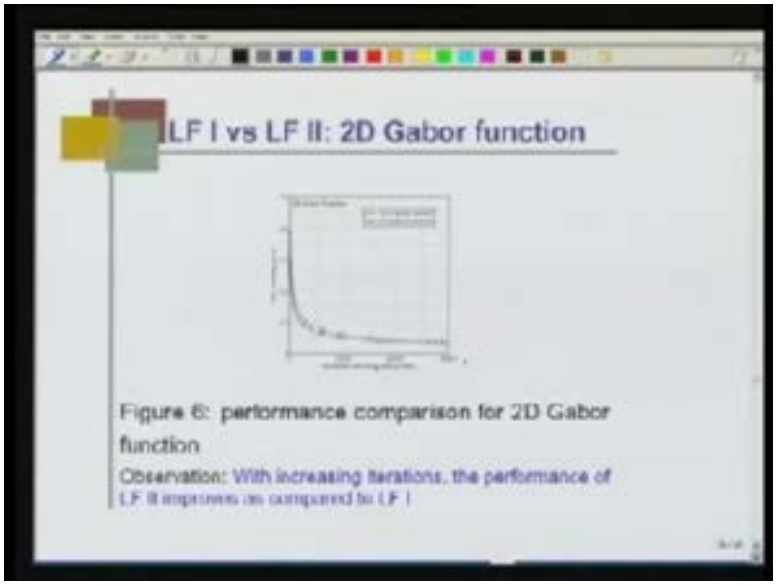
The observation is LF-II provides tangible improvement over LF-I both in terms of convergence time and training epochs and the most important is that it is invariant to initial condition. I have to recognize that training a neural network means you start from a new initial condition. Each initial condition will lead to a different convergence time. If that is the case, then the algorithm is not a good one. So, can I say that I have an algorithm that ensures that whatever may be my initial weight vector, my convergence time is independent of that? That is important. That was for XOR function.

(Refer Slide Time: 48:12)



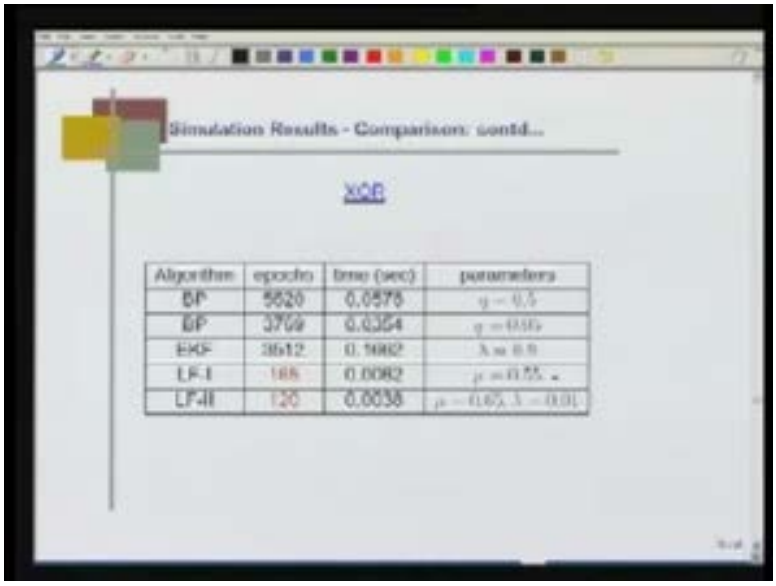

This is for three-bit parity problem and in three-bit parity problem, you can easily see…. I hope that you can get all these functions – they are there in many textbooks. This is XOR function three-bit parity. If you take a three-bit parity function, again you can see that LF-I is again very much fluctuating between 750 epochs to almost 3,000 epochs. It fluctuates, whereas in LF-II, the number of training epochs is fixed almost at the 500 level and some was fixed very rarely. Here, one fluctuation, second fluctuation here, big fluctuation 3. Again, LF-II performs better than LF-I both in terms of computation time and training epochs.

(Refer Slide Time: 49:08)




This is a 2D Gabor function. For this Gabor function, the LF-I and LF-II, if you look at the RMS error and these are the training iterations, you can easily see that for LF-II, the error is much less and it has a lesser value than the LF-I and so, it has a better convergence performance.

(Refer Slide Time: 49:42)

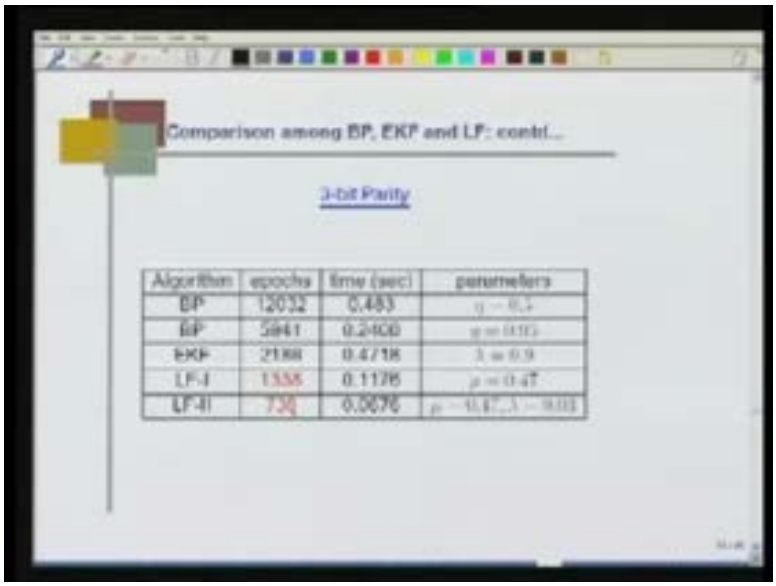



We have a kind of a table format. We have compared the results for performance of back propagation algorithm and LF-I and LF-II for an XOR function. You can easily see that the

number of epochs that are necessary for training the XOR function using eta equal to 0.5 is 5,620 and when eta equal to 0.95, it is 3,769, whereas in the case of LF-I, we take only 165 and LF-II only 120. The epoch simply gives the number of times I have to train, but what is the total computation time?

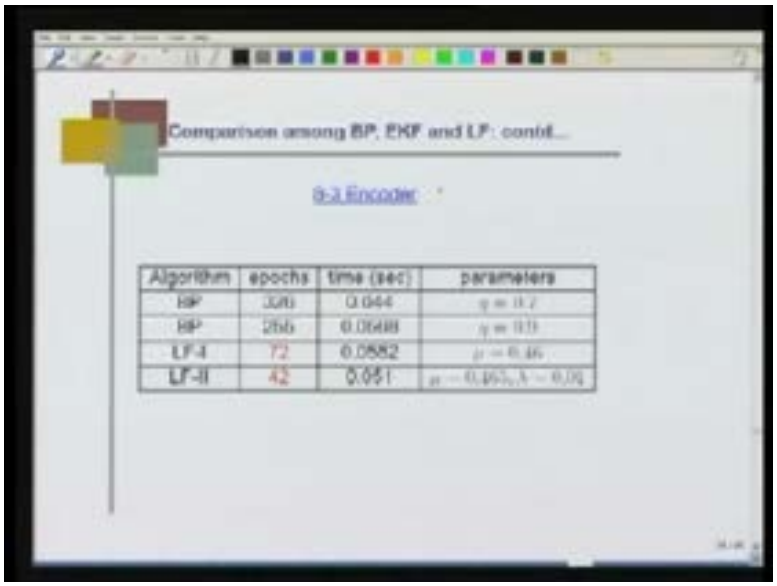
Of course, one can argue that gradient descent is a very simple function and computationally very simple but if you look at the total number of computation times in seconds that is necessary, if you again see that also, in that case, the proposed algorithm is much better because it is 0.0038 whereas for back propagation, it is 0.0354. It is pretty fast, at least 10 times faster than the back propagation algorithm in convergence time.

(Refer Slide Time: 51:10)



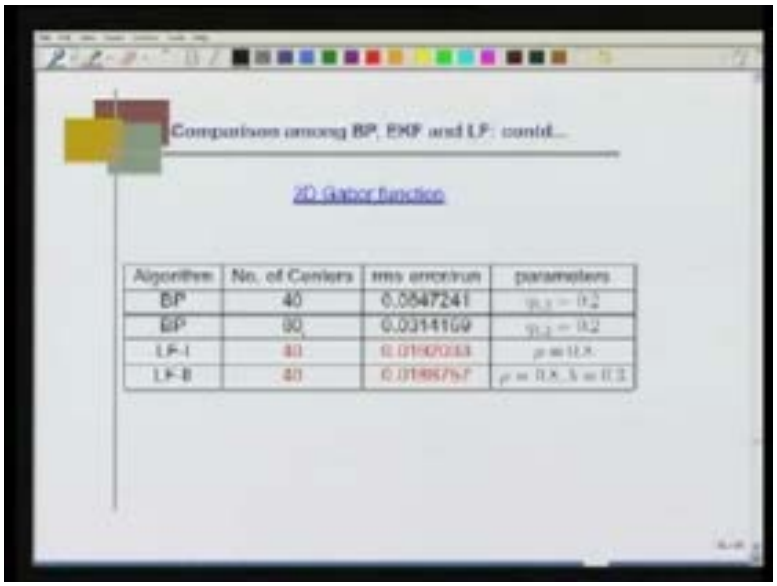

The three-bit parity problem. Again, you can easily see here that LF-II takes 738 number of epochs and LF-I is 1,338 epochs, whereas back propagation takes a pretty low time of 5,941 with eta 0.95 and eta = 0.5 is 12,032. If you look at the computation time also, for LF-II, it is 0.0676 seconds, whereas back propagation takes 0.483 seconds for training. This is the three-bit parity problem we talked about.

(Refer Slide Time: 52:17)



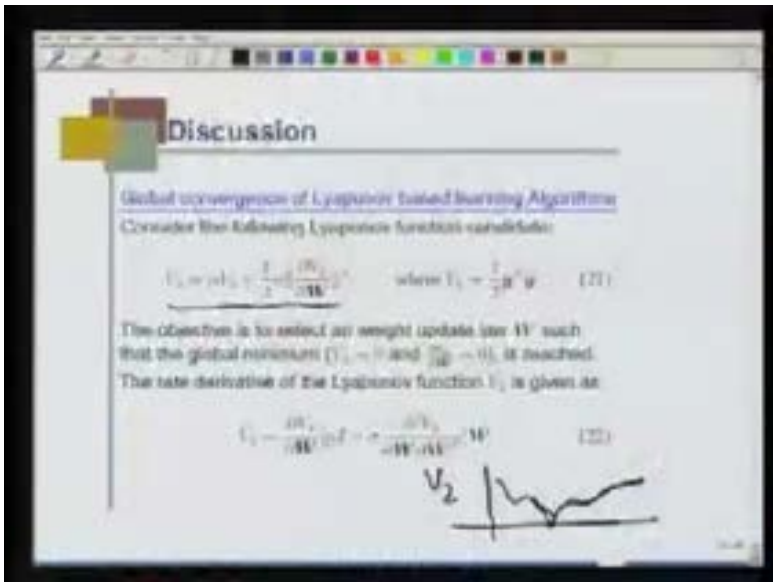

This is the 8 by 3 encoder. You already know what an encoder is. If I use an 8 by 3 encoder map using a feed-forward neural network, again if you look at the number of epochs that are necessary – LF-II and LF-II using Lyapunov function, they are pretty small compared to back propagation, and you can see the comparison time-wise also.

(Refer Slide Time: 52:19)



The 2D Gabor function. We took a multilayer network with 40 hidden units, 80 hidden units and we tried using back propagation network as well as LF-I and LF-II only 40. You can see the RMS error per run. We train the network using many initial weight conditions and average rate over 100 such runs and then we computed the average error and you can easily see the computation is in terms of RMS error. LF-I and LF-II are much better than the back propagation algorithm.

(Refer Slide Time: 53:11)

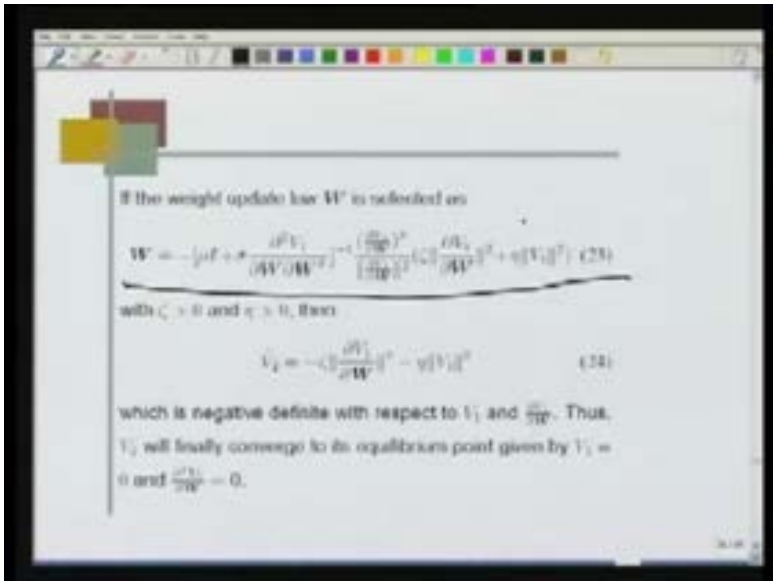



Let us come to the final part, the global convergence of Lyapunov based learning algorithm. It is possible…. If we take the objective function $V_2$ (Refer Slide Time: 53:29), Lyapunov function is mu $V_1$ plus half sigma this square, where $V_1$ is this quantity which is the Lyapunov function we have considered for our first theorem. If I compute $V_2$ dot, then this becomes equation 22. The objective is to select a weight update law W dot such that the global minimum $V_1$ equal to 0 and del $V_1$ by del W equal to 0 is simultaneously reached.

You see that if I have a cost function, then at this, del $V_1$ by del W also 0 and $V_1$, this quantity, is also 0 here. In this case, this is 0 – del $V_1$ by del W is 0, but $V_1$ is not 0 here, $V_1$ is 0 here. Let me now try to explain what the meaning of attaining global convergence is. If I look at the Lyapunov function, which is now $V_2$, you can easily see that if this is my Lyapunov function, at this point, $V_1$ is 0 and at this point, this quantity del $v_1$ by del W is also 0. $V_2$ is 0 here in this

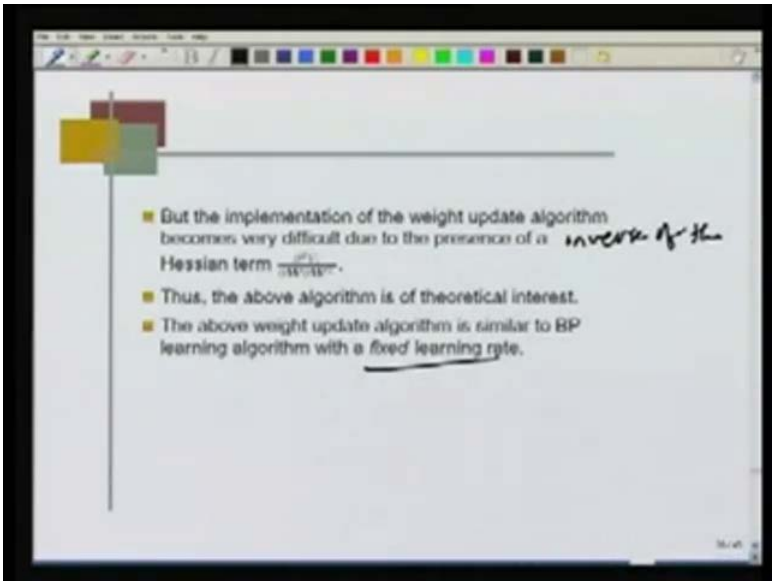
point, but in this case, this one is 0 but this is not 0 (Refer Slide Time: 55:26) in this case; in this case, this one is 0, but this is not 0. If I can take a Lyapunov function like this, some researchers have proved that we can attain global convergence. But what is the problem if I take 22? This has a Lyapunov function, which is 21. Then, I have to make sure that this $V_2$ dot is negative definite.

(Refer Slide Time: 55:56)



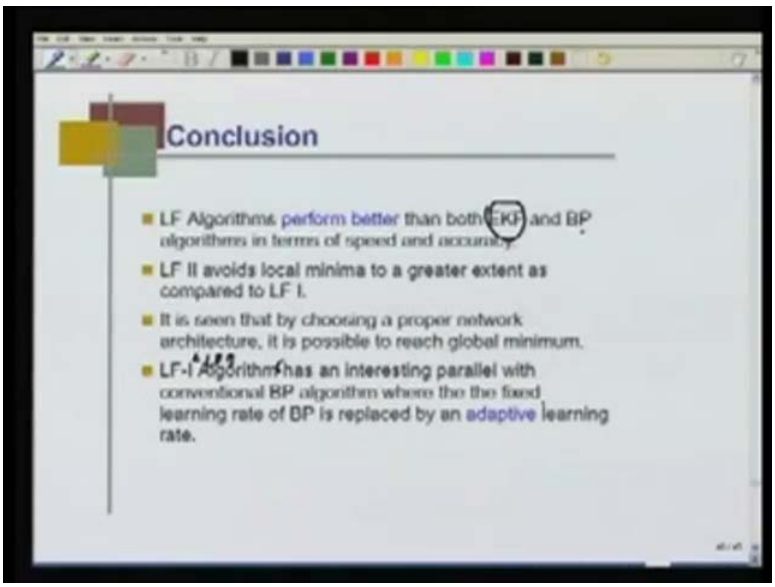

For that, you see that this is my weight update law – researchers have shown it, but unfortunately, this involves a Hessian matrix and its inversion. Computationally, it is impossible. Normally, a neural network will have a number of weights, at least in 100s. In a tangible application, this number is at least 100. In that case, you can easily see that this is a very huge matrix whose inversion is very difficult. It is not difficult to compute the inversion, but it is time-consuming. Obviously, they cannot be applied online, but the algorithm that we presented today is applicable online.

(Refer Slide Time: 56:47)



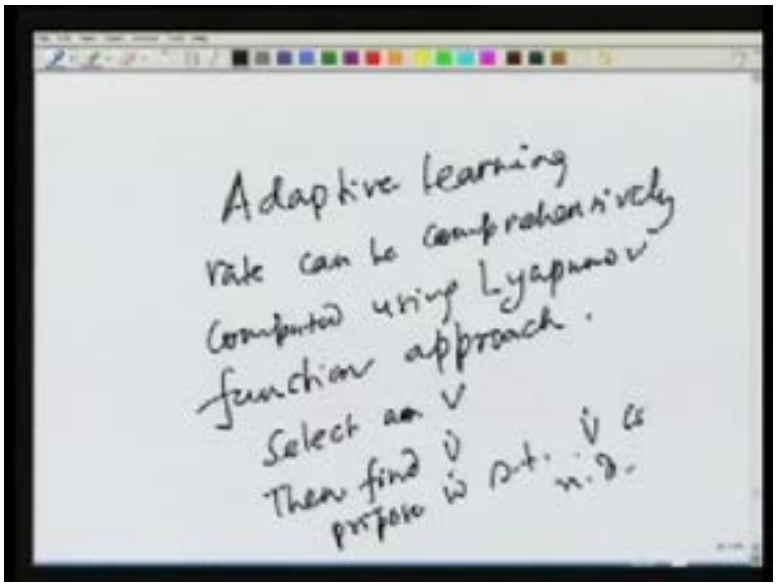

The implementation of weight update algorithm becomes very difficult due to the inverse presence of the inverse of the Hessian term. The above algorithm is of theoretical interest only and also, this weight update algorithm is similar to BP learning algorithm with a fixed learning rate, whereas the learning rate that we presented today is adaptive.

(Refer Slide Time: 57:18)



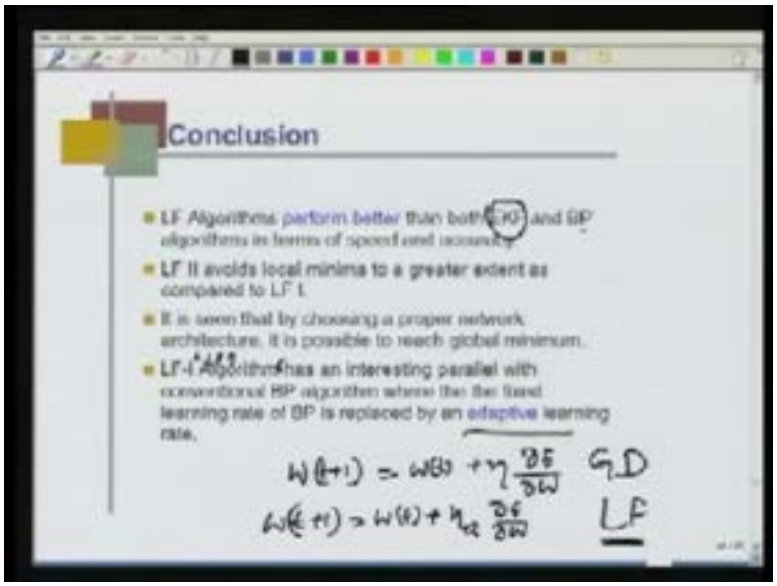
The Lyapunov function algorithms performs better than… (I did not talk about today what is extended Kalman filtering, it is not necessary) performs better in comparison to back propagation algorithm in terms of speed and accuracy. LF-II, the second algorithm that we proposed today, avoids local minima to a greater extent as compared to LF-I. It is seen that by choosing proper network architecture, it is possible to reach global minimum. Both LF-I and LF-II have an interesting parallel with conventional BP algorithm, where the fixed learning rate of BP is replaced by adaptive learning rate.

(Refer Slide Time: 58:10)




The conclusion is that the adaptive learning rate can be comprehensively computed using Lyapunov function approach. Here, we select a Lyapunov function V, then find V dot and propose W dot such that V dot is negative definite.

(Refer Slide Time: 59:09)




By doing this, what we did is that we attained an adaptive learning rate. In gradient descent, we have W t plus 1 is W t plus eta del E by del W. This is gradient descent. In Lyapunov function, we got the same plus $\text{eta}_{\text{adaptive}}$ del E by del W. We derived the Lyapunov function in adaptive eta – the structure is the same otherwise. I hope that this is clear for you – how to compute comprehensively an adaptive learning rate such that our convergence is better. Although the two algorithms that we proposed do not guarantee global convergence, they give us an idea how we can certainly improve the convergence of such algorithm for feed-forward networks. Thank you.