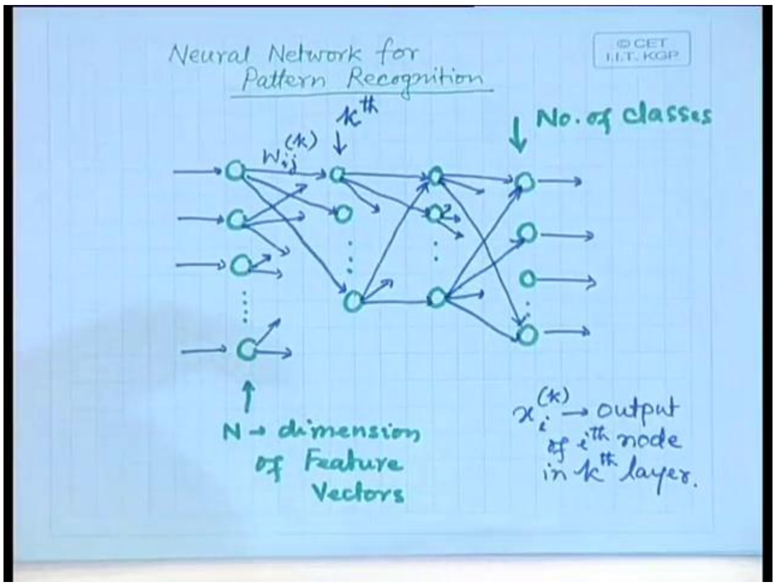
Prof. P. K. Biswas
Department of Electronics and Electrical Communication
Indian Institute of Technology, Kharagpur

Lecture - 26
Neural Networks for Pattern Recognition (Contd.)

Good morning. So, in the last class we have started our discussion on multilayered neural network or multilayer perceptron, which is used for pattern recognition purpose. And we were talking about how to adjust the vectors or how to train the neural network, so that it is ready for the recognizing the unknown feature vectors. So, what we were talking about is something like this.

(Refer Slide Time: 00:47)

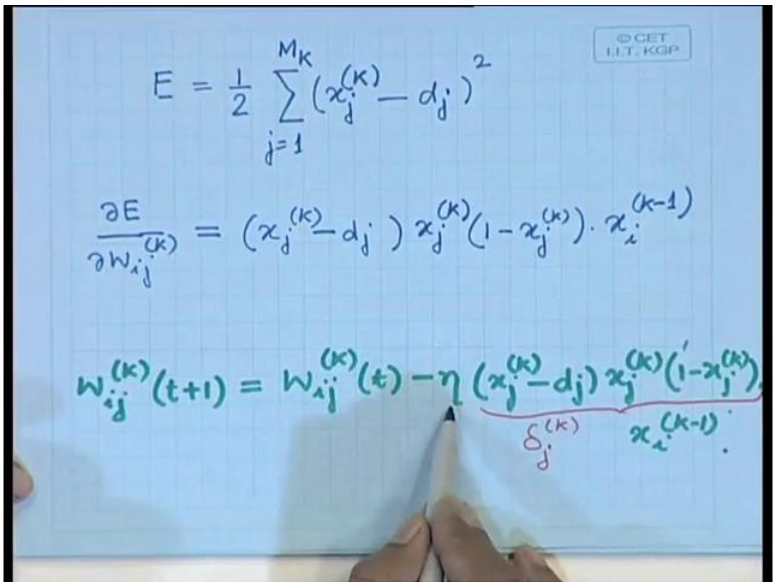

Neural Network for Pattern Recognition
© CET I.I.T. KGP
$W_{ij}^{(k)}$
k^{th}
No. of classes
N → dimension of Feature Vectors
$x_i^{(k)}$ → output of i^{th} node in k^{th} layer.

So, we have multilayered neural network with a number of input nodes, and the network has a number of output nodes, but the number of input nodes is same as the dimensionality of the feature vectors. So, it is dimension of feature vectors and at the output we have say the number of nodes which is same as number of classes. And we have 1 or more hidden layers and hidden layers will also have a number of nodes, and typically as we said the number of hidden, if we have say around 2 hidden layers. So, total we have 4 numbers of layers, including the input layers and output layer that is mostly sufficient for most of the practical applications.

And what you do is, you feed the feature vector to the input layer and get the output from the output layers and every node of the input layer is connected to, is feeding the input to every node in the layer above it. So, it goes like this, with this similarly, here. So, if I consider a kth layer node, the output of the ith node, in the kth layer is represented by x i k. So, this is the output of ith node in kth layer and the connection weights from an ith node in k minus first layer to, a jth node in the kth layer is represented by, the w i j k. So, given this and as we said that pertaining the neural network as we are inputting all the training feature vectors and we know that, what is the class belongingness of the training feature vector.

So, from that node is I know that, if I feed a known vector or training vector as input to the neural network, I know what we a corresponding output, because for every training vector, its class belongingness is known. So, if I get any vector from the output, which is not same as the expected output then the difference between these two vectors is the error vector. And from that we compute, the squared error and training of the neural network aims to adjust the with vectors in such a way that, the some of squared error that is minimized. So, what we have assumed is say, we have capital K number of layers so that, the output layer is labeled as capital Kth layer.

(Refer Slide Time: 05:05)



$$E = \frac{1}{2} \sum_{j=1}^{M_K} \left(x_j^{(K)} - d_j\right)^2$$

$$\frac{\partial E}{\partial w_{ij}^{(K)}} = \left(x_j^{(K)} - d_j\right) x_j^{(K)} \left(1 - x_j^{(K)}\right) \cdot x_i^{(K-1)}$$

$$w_{ij}^{(K)}(t+1) = w_{ij}^{(K)}(t) - \eta \underbrace{\left(x_j^{(K)} - d_j\right) x_j^{(K)} \left(1 - x_j^{(K)}\right)}_{\delta_j^{(K)}} x_i^{(K-1)}.$$

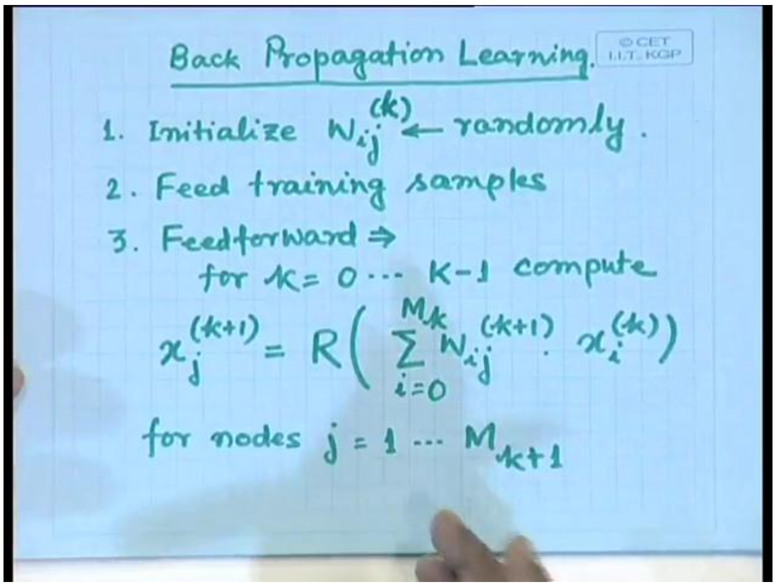
So, given that the sum of squared error is defined to be half then x j K, where this K is the capital K, indicating that this K is the output node minus dj that is the target output, expected output of the jth node in the kth layer. Take the square of this, where this j will vary from say 1 to MK, where MK is the number of nodes in the Kth layer. And we have seen that for a designing or for deciding, adjusting the weight vectors we will follow the gradient decent approach and have to take the derivative of this, with respect to w ij K. And using that derivative we have to for, adjusting the weight vector following the gradient descent approach.

So, what we have done is taken the gradient of this, del E upon del capital w ij K and we have seen that, this derivative comes out to be a x j K minus dj into x j K into 1 minus x j K into xi, from layer K minus 1. So, this is what we have seen in the last class and following this, the weight updation rule, will be of this form w ij K in iteration say, t plus 1 from the weight vector in iteration t. This will be simply, w ij K t minus eta times x j K minus dj minus x j K into 1 minus x j K times minus 1. This is the weight updation rule. Now, over here, for simplicity of expression, this particular term x j K minus dj into x j K into 1 minus x j K, this we can represent as delta j k.

So, our weight updation rule will be w ij K in iteration t plus 1, will be w ij K in iteration minus eta times delta j k into x K minus 1. So, you find that, for adjusting the weights what you do is, you start the error at the output layer, start with error at output layer, take the gradient of it with respect to the w ij K then you adjust all the weights connection weights from the K minus first layer to the kth layer. Then what ever error you get in that layer, using that you adjust the second layer, K minus second layer to K minus first layer. Then you come to k minus third layer and so on. So, we find that we have an information flow in two directions, one is from input side to output side, when you go for the computation of the outputs of different nodes.

And secondly for weight updation, what you do is to basically propagate the error term from output layer to the input layer. So, that is why, this sort of network is called as feed forward and back propagation network. And the corresponding neural network learning is called as back propagation learning. So, your vector information of input, vector information or output computation proceeds, from the input layer to the output layer. Whereas, while training the weight updation that proceeds from the output layer to the input layer that is, in the backward direction. So, this is called feed forward, but back propagation network, so accordingly the learning that is also called back propagation learning.

(Refer Slide Time: 10:26)



Back Propagation Learning.

1. Initialize $W_{ij}^{(k)} \leftarrow$ randomly.
2. Feed training samples
3. Feedforward $\Rightarrow$
   for $k = 0 \cdots K-1$ compute

$$x_j^{(k+1)} = R\left(\sum_{i=0}^{M_k} W_{ij}^{(k+1)} \cdot x_i^{(k)} \right)$$

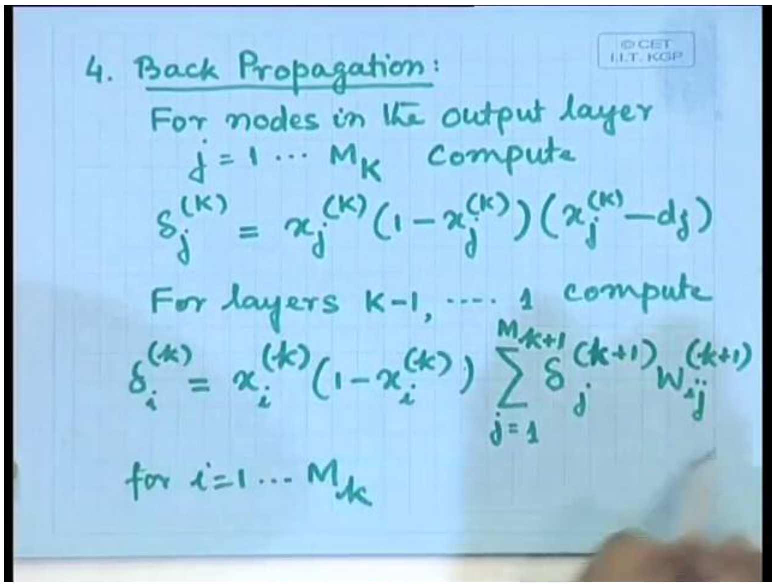
for nodes $j = 1 \cdots M_{k+1}$

So, let us see that, what will be the steps in the backward propagation learning. Obviously in the first step, I have to initialise w ij K for every weight vector, for every vector connection, for every node i, for every node j and for every node K. And this initialization, has to be made randomly so say it or initialize the weight vector w ij K randomly.

So, after we initialize the weight vectors, the next step will be that, I have to feed training vectors or training samples to the input layer. So, once I feed the training samples, then using these training samples and whatever the random weights, that has been initialized, using that I have to compute the outputs from all the nodes, from all different layers. So, the next step, will be the feed forward class and in feed forward class, I have to compute the outputs of every neuron in every layer. So, for the k is equal to 0 to capital K minus 1, I have to compute, 1 I have to compute x j K plus 1, which is nothing but that non-linear function of sum of w ij k plus 1 times x i k at the summation will be from I equal to 0 to a Mk, where Mk plus 1 is the number of nodes in the kth layer because one of the node will be used to take care of the ((Refer time: 13:42)). So, this I have to compute for every node, j equals to Mk plus 1, one of the other nodes will have a constant input.

So, this is what I have to do for in the feed forward cross, where I am computing the outputs from every node, in every layer depending upon, the weights that I have till that time and what is the input vector ((Refer time: 14:25)). So, once all these outputs are computed. Now, you come to the output layer node at the output layer node, I know what is the target output and what is the output that is computed to from here. So, from these two, I will get the error and using this error. Now, you follow the back propagation part, the proposition procedure to keep on adjusting the weights so that, error is minimized.

(Refer Slide Time: 14:55)



4. Back Propagation:

For nodes in the output layer

$j = 1 \cdots M_K$ compute

$$\delta_j^{(K)} = x_j^{(K)}(1 - x_j^{(K)})(x_j^{(K)} - d_j)$$

For layers $K-1, \cdots 1$ compute

$$\delta_i^{(k)} = x_i^{(k)}(1 - x_i^{(k)}) \sum_{j=1}^{M_{k+1}} \delta_j^{(k+1)} W_{ij}^{(k+1)}$$

for $i = 1 \cdots M_k$

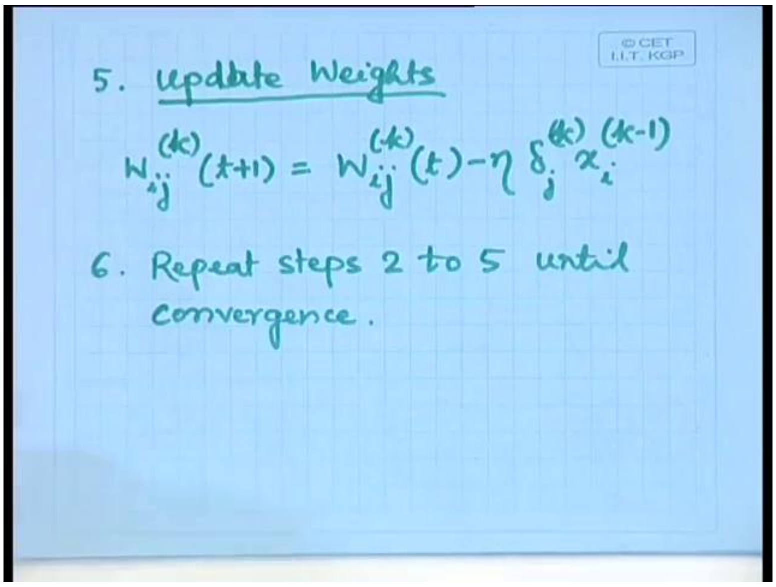
So, the fourth step will be the back propagation part. So, in the back propagation it is first we have to consider the output molecular nodes. So, for nodes in the output layer, that is j is equal to 1 to MK, where K this K is capital. I have to compute delta j, which is equal to x j K into 1 minus x j K into x j K minus dj, which is exactly same as this term, x j K into 1 minus x j K into x j K minus dj. So, this is what you do for the nodes in the output layer, but you find that computation of this term, this increment term for input layer nodes, for the hidden layer nodes has to be slightly different, because there the contribution to error is because of propagation from various other nodes.

So, for other nodes, for layers the K minus 1 up to layer 1, this computation will be slightly different. So, here we say, delta i k that will be is equal to x i k into 1 minus x i k then I have to have this summation term because I have the continuation of various nodes leading to this particular error. So, this will be delta k plus 1 into w ij k plus 1, but this j will vary from 1 to Mk plus 1. And this have to compute for i is equal to 1upto Mk, where this k is lower case.

Student: Even for the output layer ((Refer time: 18:13))

At the output layer, I can directly compute there because I know what is the target output, but in case of hidden layers. I do not know what is the target output from hidden layers, because that is coming due to contribution from various other nodes. So, this direct computation, that have done at the output layer that is not possible in the hidden layers. I am not showing this derivation of this, but following the summation from various outputs and taking the derivative of this, you can find out that this delta Mk will come out to, can be of something like this. Because at the output layer, I know what is the target output, but in the hidden layers I do not know what the target output because it is getting combined from various nodes, from the various nodes from the layer below that. So, once I have this delta terms then comes the weight updation part.

(Refer Slide Time: 19:16)



5. Update Weights

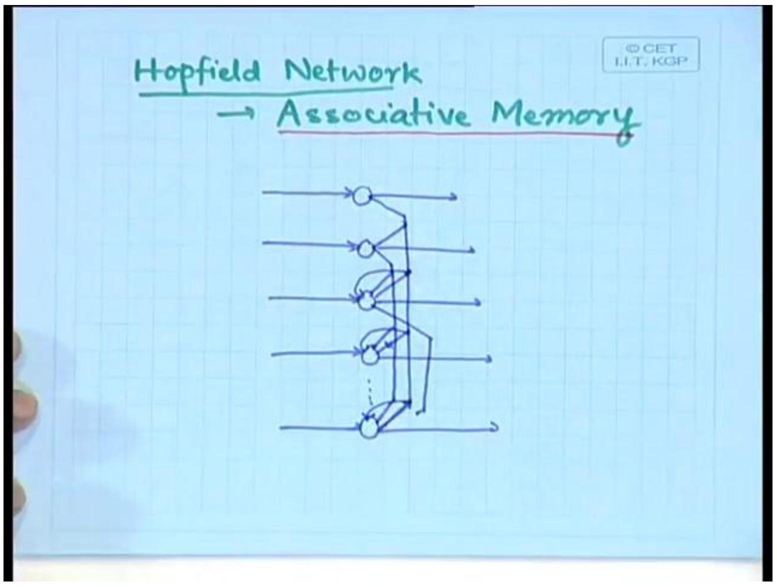
$$W_{ij}^{(k)}(t+1) = W_{ij}^{(k)}(t) - \eta\, \delta_j^{(k)} x_i^{(k-1)}$$

6. Repeat steps 2 to 5 until convergence.

So, I have to update the weight and this weight updation is simply w ij k, I will put it as t plus 1, that will be equal to w ij k t minus eta times delta j k into x i k minus 1. So, why I have put this delta term is that, having different computations of this delta value, I can have a single expression for weight updation. And all these steps are to be repeated, so sixth will be repeat steps 2 to 5 until, convergence. So, what is our condition for convergence? The condition of convergence will be either all the training samples are correctly classified without the updation of the weight vectors, without the updation of the connection weights, in a single process. Or the output error that I get, the output error is limited, it is up to a ((Refer time: 21:14)) level.

So, if one of these two conditions is satisfied then we can say that the algorithms are conversed and whatever the connection weights we have at that point of time, the network has to work with those connection weights. So, that is what is the back propagation neural network and you find that the back propagation neural network, works with the continuous inputs that means, the input vector can be of any form, it can be a desperate output vector or a continuous output vector. There is another kind of neural network, which is also used for ((Refer time: 22:00)) purpose and that neural network can be used for, recognizing the binary patterns.

(Refer Slide Time: 22:10)



Hopfield Network

→ Associative Memory

So, this is what is called Hopfield network or this also works as the weight works or something like an associative memory. Now, what is this associative memory, to find that suppose you have met one of their friends, one of your school level friend, at this age. Possibly after your 4th standard or 5th standard, you have not met him at all and you find that during this period, there is been lot of changes. So, some Thomas who was your classmate, when you were in 4th standard, it is not the same Thomas, when he was aged around 22 years old. There has been lot of changes in him, but still you are able to recognize him, may be that you have forgotten his name that is ok, but still you can say that I know this person, I have met him somewhere before.

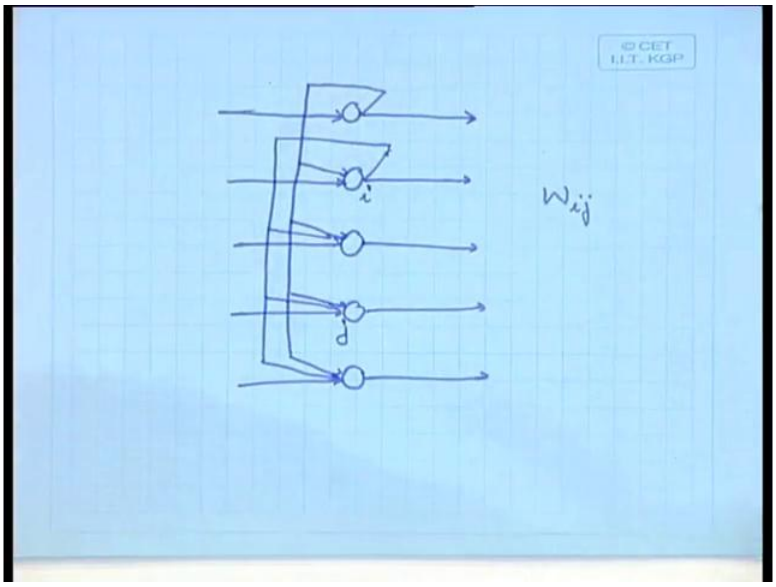
So, some sort of association you can do with some pattern that is stored in your memory. So, that is what is associative memory, so given a pattern even if the pattern is not perfect, it is an imperfect pattern still, if you have a pattern stored in the memory and wants an imperfect pattern, imperfect form of the same pattern, or distorted form the same pattern is presented. Then you can recollect from the memory, the perfect pattern which has been stored, which is very close to this imperfect pattern. So, that is what is the concept of associative memory, I can associate an imperfect pattern with a perfect pattern, which is stored in memory.

So, here we are assume that the pattern vectors or the feature vectors that we have are binary feature vectors and whenever I have a binary feature vector, the, by binary, what we means is the different feature components will be 0's and 1's. So, in this particular case, everyone will be represented by plus 1 and 0 will be represented by minus 1, but still it is point because only two digits, one corresponding to plus 1 and the other one is minus 1. So, the feature vectors that we have a binary feature vectors and suppose there are n number of components of the binary feature vectors, that means I have n ((Refer time: 25:17)) patters.

Now, the difference between the back propagation network or multi-layer perceptron and this case Hopfield neural network is that, in case of back propagation network, your information flows only from the input layer to the output layer. A node in the k minus first layer will always feed to a node in the kth layer, a node in the kth layer will always feed to a node in the k plus first layer, but a node in the kth layer does not give information, does not feed a node in the k minus first layer. So, that back propagation or backward information flow, only we use for training the network, but not for computation. In case of Hopfield network, output of every node is connected to the input of every other node.

So, it is something like this, see if I have n number of nodes, every node obviously has an input that is a binary pattern. They have outputs and output of every node is connected to the input of every other node. And this connection is bi direction that means, information flows in both the directions. So, similarly from here, this goes to the input side not from the output side, this connection will go to the input side.

(Refer Slide Time: 27:47)



Let me draw a fresh figure. So, like this output of every node is connected to the input of every other node and as before, the output of ith node is connected to the input of the jth node via a connection with w ij. And this is the same connection with, which is used for connecting output of the jth node to the input of ith node. So, wij and wji for same i and j will be same and what you do is by training as before, here what we are doing is, we are actually storing a number of patterns. A number of binary patterns in this associative network, or in this Hopfield network by adjusting the weights property. So, it is the weights wij, which actually stores the patterns and when you feed an input, because it is a feedback network from every node, I get input to, I get input to every other node.

So, it is the feedback network and all the notes can be fired or can be activated asynchronous that is, in which order the nodes actually operate that is not specified. They can work in any order, so finally when you feed an input there will be a number of iterations, which occur asynchronously among all the nodes. And finally, when the nodes stabilizes, when the network stabilizes the output vector that in, that will get it is one of the vectors, which is already screwed in the network, by adjusting the connection weights, which is closest to the input vector, that we have given. So, when obtain the network we obtain a network in the network using the perfect feature vectors, perfect vectors and while retrievable, even if I given input vector, which is not perfect.

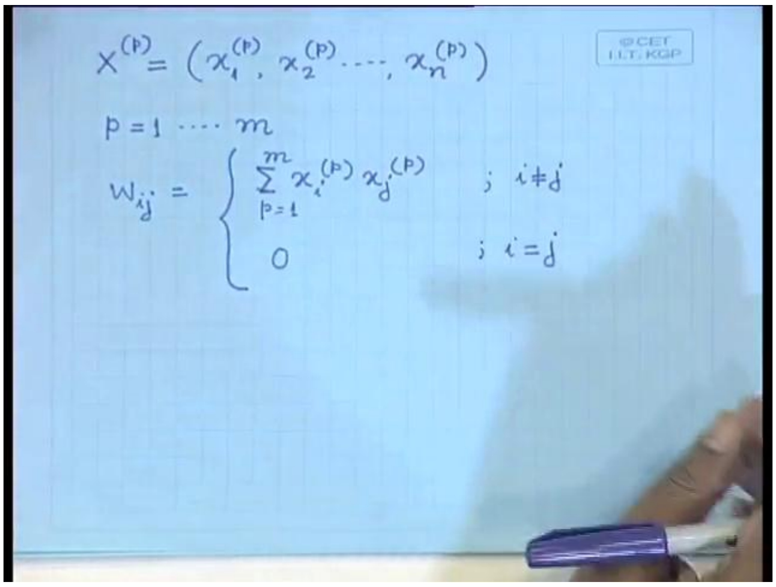
The network will retrieve one of the patterns, which is perfect and stored and closest to the imperfect input vector that we have presented. So, that is how this Hopfield neural network works and while changing the output, while retrieving a particular pattern the output of a network will be set to class one or it will be positive, if some of the weighted imports is greater than 0 and it will be negative, if it is not greater than 0.

So, for less than or equal to 0, the output of the node will be negative, for weighted input, some of weighted imports if it is greater than 0, the output is positive. So, you will find that during iteration the output of a node can vary many times, before it actually stabilizes because every change in output of one node is deflected to the input of every other node.

And as it is deflected to the input of every other node, so our change of output of one of the nodes may effectively change the output of some other nodes. And that change again may reflect the output of the previous node because this change is again deflected to the input of the previous node. So, the network will go through a number of iterations, iteratively and during the iterations output of the notes may vary from plus to minus from plus to minus many times, before the network actually stabilises.

And when it stabilises, the output pattern that we get, that is the closest patterns, which is stored in the network and closest to the input pattern, that has been presented to this neural network. So, actually it is associating the input pattern to one of the stored patterns so that is why it is associative memory or associating network. Now, let us see that, how we can store the patterns in the weight vectors or the connection weights from the ith node to the jth node. So, when we select the weight vector w ij are to consider the, training patterns that has been given, that has been provided.

(Refer Slide Time: 33:21)


$$X^{(p)} = \left(x_1^{(p)}, x_2^{(p)}, \ldots, x_n^{(p)}\right)$$

$$p = 1 \cdots m$$

$$W_{ij} = \begin{cases} \sum_{p=1}^{m} x_i^{(p)} x_j^{(p)} & ; i \neq j \\ 0 & ; i = j \end{cases}$$

So, suppose I have this pth tending pattern Xp and as I said that this Xp of the training vectors, the sample vectors are n-dimensional vectors. So, it will have n number of components and let us say of those n number of components are x 1 p, x 2 p up to x n p. As I have n number of components in the feature vector, each of the x 1, x 2, x n they can assume either a value plus 1 or a value minus 1, because they are binary patterns and our conventioneers that bit 0 will be represented by minus 1. So, every component can have a value either plus 1 or minus 1 and this is the pth training vector, feature vector. And I have say m number of feature vectors that was this p, it can vary from 1 to m, as a have a m number of feature vector, or n number of pending vectors.

Then, using this training vector, I have to compute the connection weights wij. So, wij is given by sum of the x i p into x j p, at this p varies from 1 to m for all i not and equal to 0 and it will be equal to 0, if i is equal to j. So, that is how, you set wij, so what does it mean, you find that whenever i is not equal to j, your wij will be computed by this. If I is equal to j then wij will be equal to 0, you find that when I have from the schematic of this Hopfield network. I have not feedback to the input of the same node. So, that means wij for i equal to j is equal to 0 means, that a node does not feedback the signal to itself.

The output of a node is connected to the input of every other node, where this connection weights wij, but output of a node is not connected to the input of the same node. So, that is what it means, and from here you find that, what is this term, ij, x i p into x j p you find that, both these components are coming from the same feature vector p. And the feature vectors or binary feature vectors having values plus 1 or minus 1, so this simply says, that if x i p and x j p both of them are same, both of them are positive or both of them are negative.

Then, this term will be equal to plus 1, if one of them is positive other one is negative then this term will be minus 1. So, overall this summation sum x i p into x j p, so this simply tells you that the number of times a bits are same, minus the number of times bits are different. So, that is how and that is computed over, all the training patterns because it ((Refer time: 37:25)) it is computed over, all the training patterns and that is stored as wij, the connection between the ith node and the jth node. So, let us take a very small example.

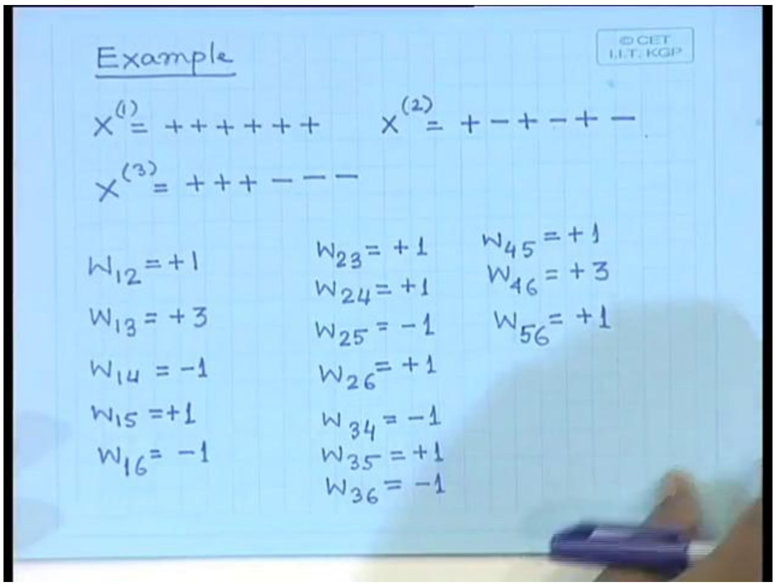
Student: ((Refer time: 37:40))

No, there is no such rule I did not put it intentionally because I said that it is the same connection, which is used for bidirectional communication.

Student: Actually, in that diagram, second one should be connected ((Refer time: 38:03))

It should be there, output of the second one should be connected to the first one, from second one to the second one there should be maintained connection, from first one to the first one there should be no connection. But the connection weight we can understand wij is named as wji ((Refer time: 38:35)).

(Refer Slide Time: 38:39)



Example

$X^{(1)} = + + + + + +$ $\quad X^{(2)} = + - + - + -$

$X^{(3)} = + + + - - -$

$W_{12} = +1$
$W_{13} = +3$
$W_{14} = -1$
$W_{15} = +1$
$W_{16} = -1$

$W_{23} = +1$
$W_{24} = +1$
$W_{25} = -1$
$W_{26} = +1$
$W_{34} = -1$
$W_{35} = +1$
$W_{36} = -1$

$W_{45} = +1$
$W_{46} = +3$
$W_{56} = +1$

So, let us take a very small example to see how this weight vectors are really assigned. So, let us take that I have say, three feature vectors, one is x 1 and this feature vectors is given by plus, plus, plus a six bit feature vector. The second one x 2, which is given by so plus 1, minus 1, plus 1, minus 1, plus 1, minus 1 and the third feature vector is x 3, which is given by. So, all the three are six bit feature vectors. So, I have to compute the different connection weights, so what will be the value of the w 12, so here you find that I have to consider the first feature vector and the second. So, from the first feature vector both of them are positive.

So, when I compute this term x i p into x j p, where i is equal to 1, j is equal to 2, that will be simply x 1 1 that is plus, x 2 1 that is also plus. So, I get product to be positive, coming to the second one, one bit is positive the other bit is negative, so it is minus. Coming to the third one, both of them are positives so I will get plus 1. When I summed over all these three patterns for p varying from 1 to 3, my result will be equal to 1. So, this w 1 2 this will be equal to plus 1 similarly, and w 12 will be same as w 21 that you can easily verify., When I compute w 13, I had to consider the plus 1 and third bit, over here it is plus 1, over here it is also plus 1, over here it is also 1.

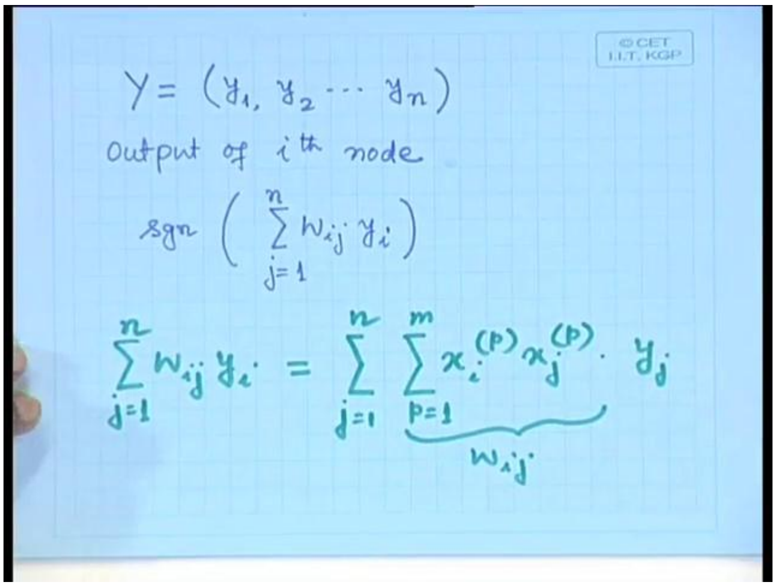
So, w 1 3 I get that is nothing, but plus 3, what is w 14 here it is plus 1, here it is minus 1, here also, it is minus 1, should effectively I get the w 14 to be equal to minus 1. Then w 1 5 here it is plus 1, here it is plus 1, here it is minus 1. So, it will be 1 similarly, w 1 6 that will be plus 1, minus 1, it will be minus 1, this computation. So, likewise you will find that, I have to compute w 2 3, I would not compute w 2 1 because the w 2 1 is same as w 1 2, w 2 2 is equal to 0. So, I have to compute w 2 3 so second and third here it is plus 1, here it is minus 1, here it is plus 1. So, 1 plus 1 minus 1, w 2 3 becomes plus 1. Similarly, w 2 4 take the second and the fourth bit, here it is plus 1, second and fourth bit here also it is plus 1 because both the bits are negative.

Second and fourth bit, it is minus 1, so that gives me w 24 also equal to plus 1. Similarly, w 2 5 second bit and fifth bit plus, plus 1 here, second bit and fifth bit minus 1 here, second bit and fifth bit minus 1 here. So, I get overall minus 1 similarly, w 2 6 here it is plus 1, here it is plus 1, here it is minus 1. So, w 2 6 is plus 1, now when I compute w 3, w 3 1 is same as w 1 3, I do not have to be compute, w 3 2 is same as w 2 3 I do not have to be compute, w 3 3 is equal to 0. So, I have to compute w 3 4, w 3 4 here it is plus 1, 3 4 here it is minus 1, 3 4 here it is minus 1. So, w 3 4 becomes minus 1, w 3 5 it is plus 1, here also it is plus 1, so here it is minus 1, so w 3 5 becomes plus one 1 w 3 6 here is plus one, here it is minus 1, her also it is minus 1.

So, w 3 6 becomes minus 1 similarly, I have to go for the w 4 5 because others I need not compute. So, w 4 and 5 plus 1 minus 1, here it is plus 1, so w 4 5 becomes plus 1, w 4 6 plus 1 minus 1 plus 1. So, this becomes plus 3 then I have to compute w 5 6, because others I need not compute, w 5 6 plus 1 minus 1 plus 1 so it effectively becomes plus 1. So, these are all the weight vectors that I compute, which stores these three different patterns. So, once I have this weight vectors, then using this weight vectors.

Now, if I feeding an unknown feature vectors, after iteration when this network will converge it will give me an output vector, which is close to the output vector, will be one of the three. And that will be closest to the input feature vector, that you have presented even if the input feature vector is not one of this. So, it will try to extract one of the stored patterns, which is closest to the input vector. Now, the question is why should this network work, how does it work.

(Refer Slide Time: 46:24)



$$Y = (y_1, y_2 \cdots y_n)$$

Output of i^{th} node

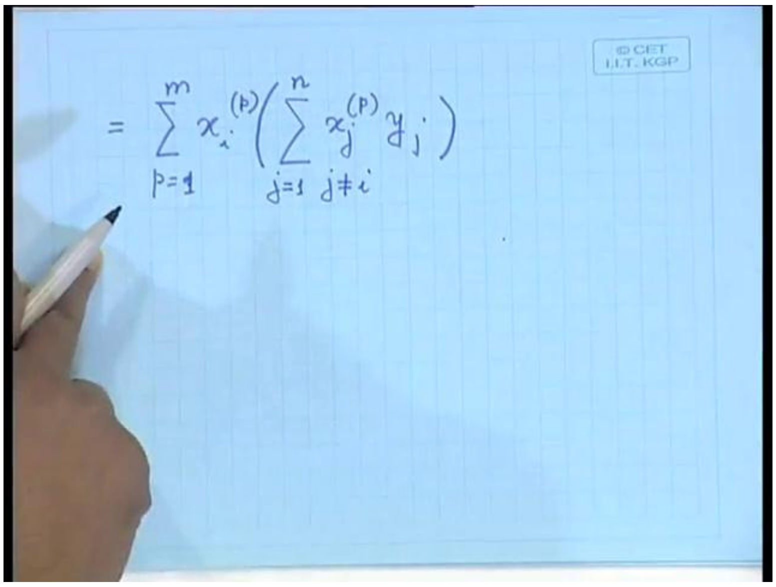
$$\operatorname{sgn}\left(\sum_{j=1}^{n} w_{ij} y_i\right)$$

$$\sum_{j=1}^{n} w_{ij} y_i = \sum_{j=1}^{n} \underbrace{\sum_{p=1}^{m} x_i^{(p)} x_j^{(p)}}_{w_{ij}} \cdot y_j$$

Suppose, i feed in an input feature vector, binary feature vectors, which is given as Y, which is nothing but y1, y2 up to say yn. It should also have the, same number of components of and given this, the output of ith node, it will be either plus 1or minus 1. And this output will depend upon, you have to compute wij times yi for, j varying from 1 to n. And depending upon the sign of this, if it is greater than 0 and output of ith node will be plus 1, if it is less than 0 then output of the ith node will be minus 1, that will be repeated many times, during the iteration. And finally, it will converge, so this sign of the output of the ith node, will actually change many times not only once and that will change asynchronously.

I cannot say that, after change of the output of the first node, then only the output of the second node will change. There is no such guarantee because everything works asynchronously mentioned and all of them are working in parallel. So, you find that, if you analyze this term, that is this summation of the wij in to y i, where j varies from 1 to n, if I analyze this, this will simply be sum of what is this wij? wij is sum of x i p in to x j p, where p varies from 1 to m. If I have m number of patterns, these times y j, where j varies from 1 to n, so this is what my wij.

(Refer Slide Time: 49:29)



$$= \sum_{p=1}^{m} x_i^{(p)} \left(\sum_{j=1 \; j\neq i}^{n} x_j^{(p)} y_j \right)$$

And this term, I can write as sum of x i p into sum of x j p, yj where j varies from 1 to n and j not equal to i. And here, p varies from 1 to n, so I can write this whole from in that manner. So, it is just rewriting this expression into this form, so here you find that these value, within this xi.

Student: ((Refer time: 50:25))

I will not be written because I want the ith output, so my variable will be j. It is the ith output. So, I have to sum it over j.

Student: ((Refer time: 50:42))

It is the output of the ith neuron.

Student: Single value?

Single value.

So, if I compute for all values of i, I will be getting that. So, here we find that depending upon the closeness of this vector x and y, this is the product of xj times yj and you are taking the summation for i is equal to 1 to n, j is equal to 1 to n. Where j is not equal to i so these value will be close to n, where n is the dimensionality, if the vector xj and yj, if the values xj and yj are matching most of the time. And this will be close to minus m, if xj and yj they do not match most of the time.

So, if they match most of the time, I get a positive output and that means, that this pth bit whatever is output, that should be the sign of the pth bit, that should the signed by the ith bit. And if they do not match most of the time, then it should be the reverse. So, if you go on iterating on this, a large number of times when the network converges, the output of the network will be the one, which is closest to the pattern, which is tored into the network. So, that is the logic behind, why this network should work was as associative network. So, I will stop this lecture today, next I will continue with some other topic.

Thank you.