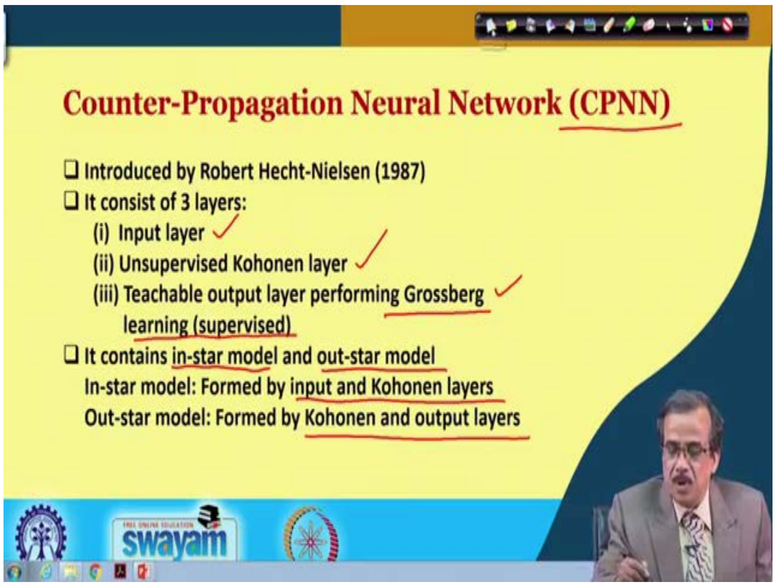
Fuzzy Logic and Neural Networks
Prof. Dilip Kumar Pratihar
Department of Mechanical Engineering
Indian Institute of Technology, Kharagpur

Lecture - 29
Some Examples of Neural Networks (Contd.)

(Refer Slide Time: 00:15)



Counter-Propagation Neural Network (CPNN)

- Introduced by Robert Hecht-Nielsen (1987)
- It consist of 3 layers:
  - (i) Input layer
  - (ii) Unsupervised Kohonen layer
  - (iii) Teachable output layer performing Grossberg learning (supervised)
- It contains in-star model and out-star model
  - In-star model: Formed by input and Kohonen layers
  - Out-star model: Formed by Kohonen and output layers

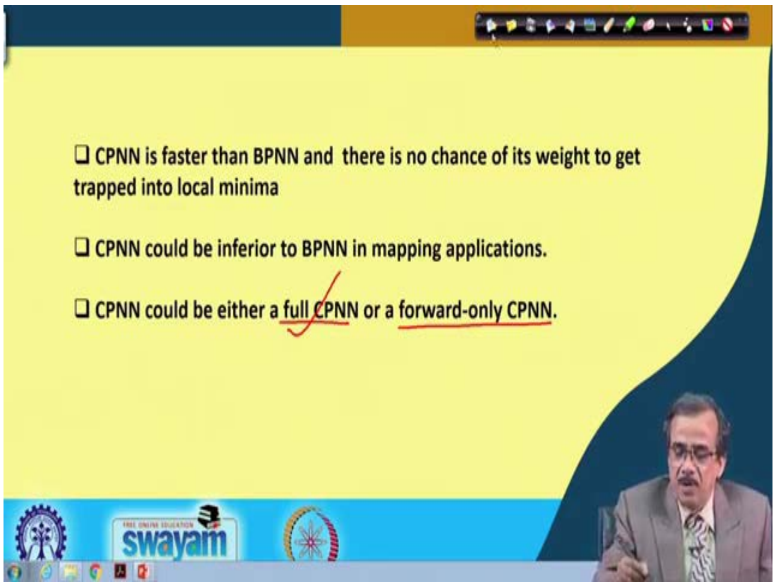
We are going to discuss the working principle of another very popular network and that is called actually the counter propagation neural network and in short, this is known as CPNN. Now, before you start with this particular network, let me tell you the purpose of developing this particular network. The purpose is once again to model the input-output relationships of a process as accurately as possible.

Now, here, this was proposed in the year 1987 by Robert Nielsen. Now, this network consists of, in fact, 3 layers one is called the input layer, we have got the unsupervised Kohonen network or the Kohonen layer and we have got actually a supervised layer, where we follow the principle of Grossberg learning. We have got one teachable output layer and this particular layer is going to perform the Grossberg learning, which is nothing, but a supervised learning.

So, in this network, we are going to consider both supervised as well as unsupervised learning. Now, let us see, how does it work? Now, construction-wise if you see, this particular network consists of two models: one is called the in-star model and another is called the out-star model. Now, this in-star model actually consists of input and the Kohonen layer and the out-star model consists of the Kohonen and the output layer.

Now, we are going to discuss, in details, what is happening in in-star model and what is happening in your the out-star model.

(Refer Slide Time: 02:21)

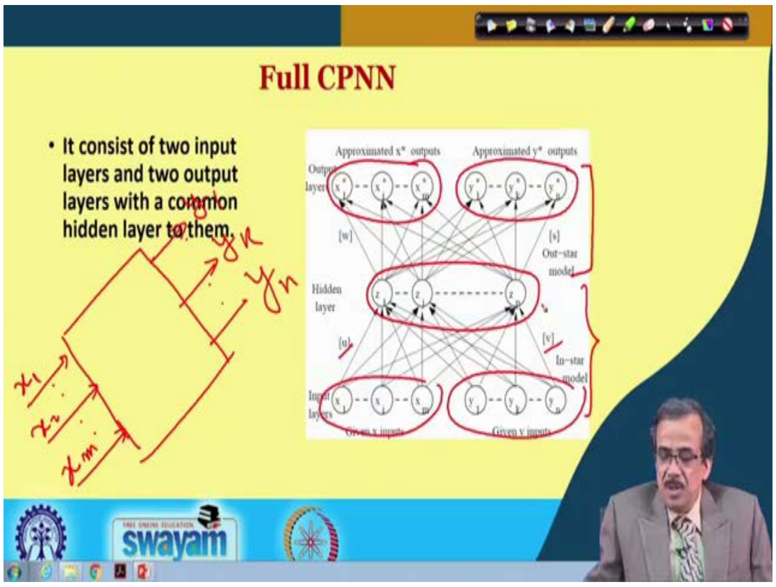


- ❑ CPNN is faster than BPNN and there is no chance of its weight to get trapped into local minima
- ❑ CPNN could be inferior to BPNN in mapping applications.
- ❑ CPNN could be either a full CPNN or a forward-only CPNN.

Now, let us see actually what is happening there, but before that let me just tell you a few facts that this particular CPNN, that is the Counter Propagation Network, is found to be faster than the multi-layered feed forward network and here, we do not use the concept of the back-propagation algorithm. So, the chance of its solution for getting trapped into the local minima is actually nil and this CPNN, if you see the performance, that is, the accuracy in prediction. So, it could be a little bit inferior compared to your multilayered feed forward network, but it is computationally faster compared to your multilayered feed forward network.

Now, if you see the literature, the CPNN could be either a full CPNN or it could be a forward only CPNN. So, we are going to discuss the working principle of both full CPNN as well as the forward-only CPNN. Now, let me first concentrate on the full CPNN and how does it work? Let us see its working principle.

(Refer Slide Time: 03:41)



Full CPNN

- It consist of two input layers and two output layers with a common hidden layer to them.

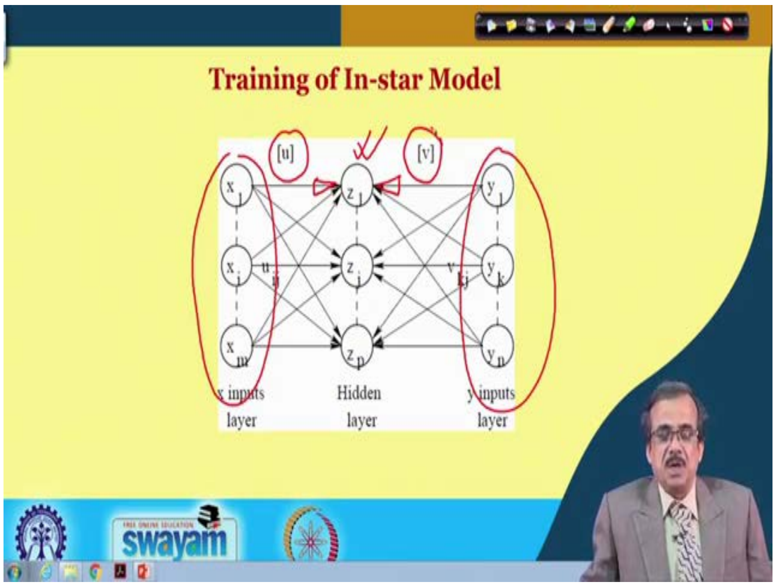
Now, this is actually the schematic view of the full CPNN and if you see construction-wise, it has got two input layers. So, I have got one input layer here. So, one input layer, here I have got another input layer here, then we have got two output layers so this is one output layer, another output layer and we have got actually a common hidden layer and this hidden layer is denoted by this. So, we have got two input layers, two output layers and one hidden layer.

Now, the in-star model consists of, as I told, the input layers and this particular hidden layer. So, this is actually, what you mean by the in-star model, and the out-star model consists of the hidden layer and the output layer. So, this is nothing, but your the out-star model and here, actually we take the help of your some correcting weights like your u then comes v, w and s and these connecting weights are lying in a normalized scale, generally we consider 0 to 1 or from – 1 to + 1.

Now, here, the purpose of this particular model is to establish the relationships between the inputs and the outputs and supposing that I have got a process and this process, I am just going to model and this process is having say n number of the inputs x_1, x_i up to say x_m. So, this is your x_1, then comes your x_i, and then comes your x_m. So, we have got m number of inputs and this process is having say small n number of outputs denoted by y_1, y_k, y_n. So, we have got your y_1, then comes your y_k and the last one is say your y_n; so we have got n number of outputs.

So, this particular process, I am just going to model with the help of your full CPNN. Now, here, actually what will have to do is, as we have already mentioned, that we will have to implement the Kohonen network fast. And, we will have to find out, who could be the winner of these particular neurons lying in the hidden layer, and then we will have to go for some sort of supervised learning, that is your the Grossberg learning and let us see, how does it work.

(Refer Slide Time: 06:43)



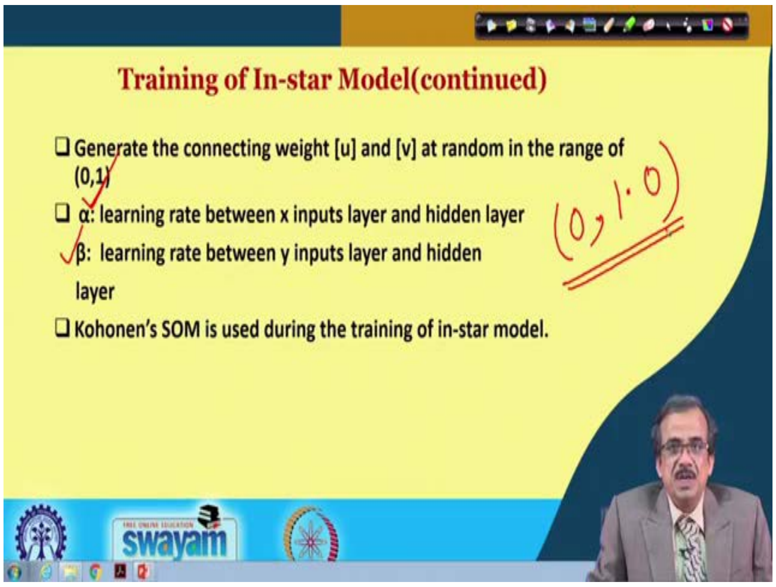
Training of In-star Model

So, let me concentrate first on this particular the in-star model and I have already mentioned that in the in-star model, we consider two input layers and the hidden layer. Now, here, so these two input layers, we are drawing in a slightly different fashion, for example, say this is one input layer, this is another input layer and this is actually your the hidden layer.

So, from this particular x, I am just moving towards the hidden and from this particular y, once again we are moving towards the hidden, you can see the directions of arrow, so these are coming from both the sides and this is nothing, but your the hidden layer and on the hidden layer, actually we have got small p number of neurons and as I told, our task will be out of this small p number of neurons, just to identify who could be the winner.

Now, the connecting weights between your x_input layer and the hidden layer is nothing, but u and the connecting weight between your y_input layer and the hidden layer is nothing, but is your the v and this is actually the construction of this in-star model, now let us see, what do you do in in-star model.

(Refer Slide Time: 08:19)



Training of In-star Model(continued)

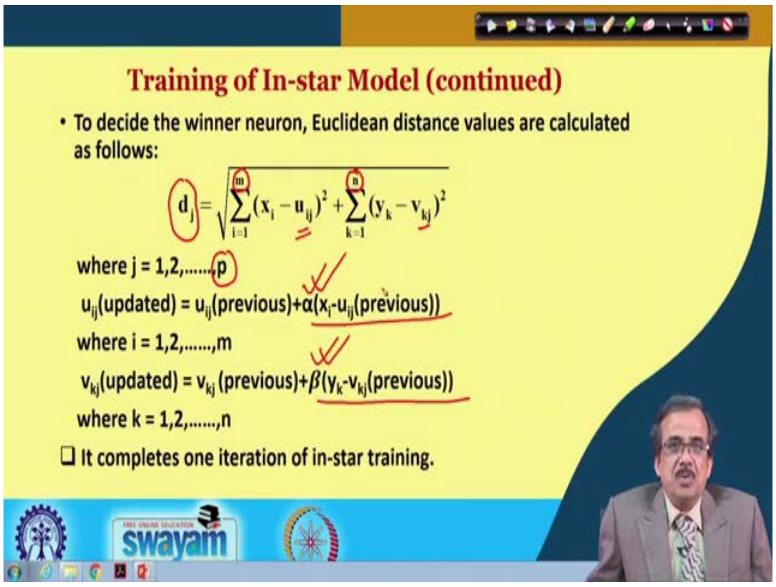
- Generate the connecting weight [u] and [v] at random in the range of (0,1)
- α: learning rate between x inputs layer and hidden layer
- β: learning rate between y inputs layer and hidden layer
- Kohonen's SOM is used during the training of in-star model.

(0, 1.0)

So, we generate the connecting weights, that is u and v in the range of say 0 to 1, and we consider some learning rate, for example, say α, the learning rate between x input layer and the hidden layer, and β is the learning rate between the y_input layer and the hidden layer, now this α and β will lie in a range of say 0 to 1. So, they are going to lie in the range of your 0 to 1 and these particular connecting weights, sorry this learning rate will vary in this range.

Now, here, as I mentioned that we are going to use Kohonen and self organizing map just to find out like, who could be the winner lying in this hidden layer.

(Refer Slide Time: 09:19)



Training of In-star Model (continued)

- To decide the winner neuron, Euclidean distance values are calculated as follows:

$$d_j = \sqrt{\sum_{i=1}^{m}(x_i - u_{ij})^2 + \sum_{k=1}^{n}(y_k - v_{kj})^2}$$

where j = 1,2,……,p

u_{ij}(updated) = u_{ij}(previous)+α(x_i-u_{ij}(previous))

where i = 1,2,……,m

v_{kj}(updated) = v_{kj} (previous)+β(y_k-v_{kj}(previous))

where k = 1,2,……,n

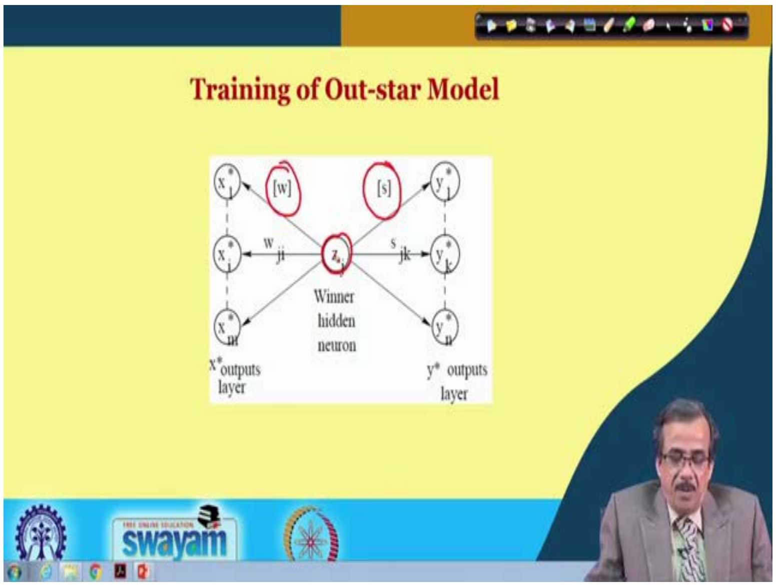
❑ It completes one iteration of in-star training.

Now, if you see the way, this particular winner is declared is as follows: So, we try to find out the Euclidean distance, that is, d_j now what is d_j? So, this is nothing but, $d_j = \sqrt{\sum_{i=1}^{m}(x_i - u_{ij})^2 + \sum_{k=1}^{n}(y_k - v_{kj})^2}$. Now, if we remember, we have got small m number of x inputs and your small n number of y inputs and this u_ij is nothing but, your the connecting weight and v_kj is nothing, but the connecting weight and here j varies from 1 to p. So, p is nothing, but the total number of neurons lying in the hidden layer.

So, for each of the hidden neurons, we try to find out, what are these d values and then, we compare and find out, which hidden neuron is going to give the minimum d value, that will be declared actually as the winner. Now, here, the connecting weights, that is, u_ij will have to be updated using this particular rule, that is, $u_{ij}(updated) = u_{ij}(previous) + \alpha(x_i - u_{ij}(previous))$. Now, here, this α is nothing but, the learning rate lying in the range of 0 to 1. Now, here, this particular i is nothing but 1, 2 up to your small m, and as I have already mentioned, so this small m is nothing but, the number of x_inputs.

Now, v_kj, this particular connecting weight, how to update? So, v_kj (updated) is nothing but, your v_kj (previous) plus β into (y_k minus v_kj (previous)). Now, this particular β is once again the learning rate. Now, once that particular process is over, now we are in a position to declare that who could be the winner lying on the hidden layer, and that completes actually one iteration of in-star training.

(Refer Slide Time: 12:21)

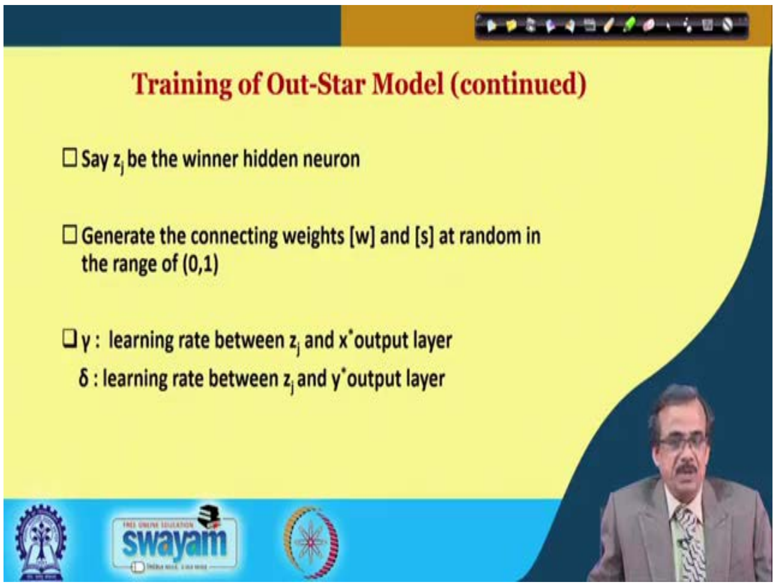


Training of Out-star Model

Now actually, we will have to go for some sort of the out-star model. Now, if you see the out-star model, the out-star model will look like this and as I told that we have already got the winner, supposing that your z_j is nothing but the winner and this particular winner is lying on the hidden layer.

Now, if you see the connecting weights, the connecting weights between this winner lying on the hidden layer and your this x^* is denoted by actually your w and the connecting weight between your the winner hidden neuron and your y^*, that is, your output layer. So, this particular connecting weight is denoted by s and we can assume or we can generate the values of the connecting weights initially at random and we can update also. Now, how to update that? We are going to discuss.

(Refer Slide Time: 13:23)

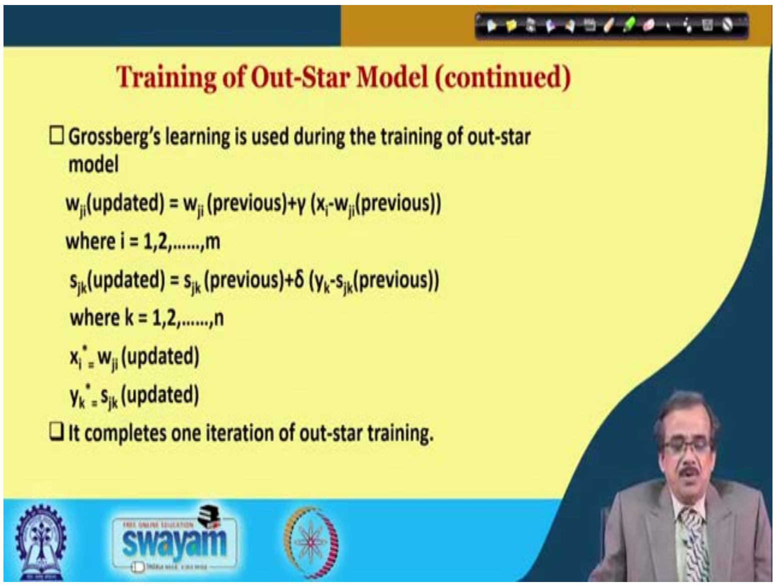


Training of Out-Star Model (continued)

- Say z_j be the winner hidden neuron
- Generate the connecting weights [w] and [s] at random in the range of (0,1)
- γ : learning rate between z_j and x^* output layer
  δ : learning rate between z_j and y^* output layer

Now, here, if you see, this I have already mentioned here, for example, z_j is the winner connecting weights lying between 0 and 1 and γ is actually the learning rate between z_j and x^* output layer and δ is actually your the output learning rate between z_j and y^* output layer. Now, if you just see so z_j and x^*, here, we have got the your learning rate and here, we have got another learning rate, now particular learning rates actually will have to assign some numerical values and their values we lie between 0 and 1.

(Refer Slide Time: 14:23)



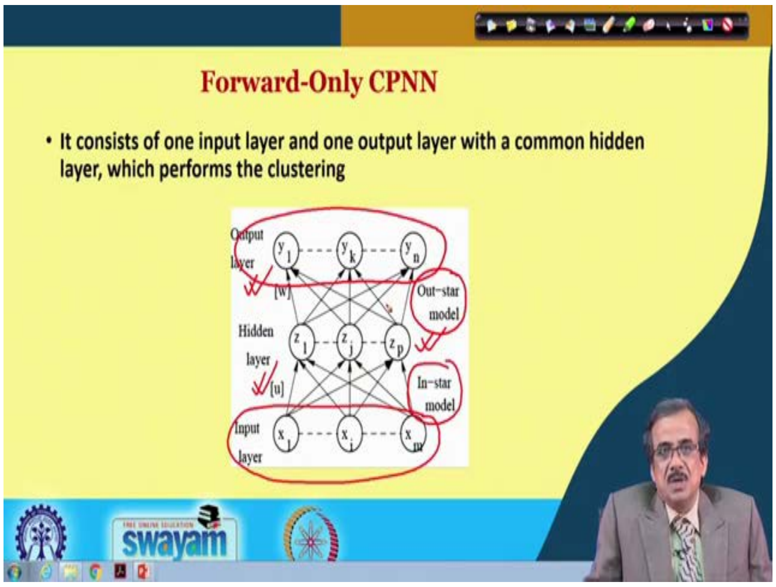
Training of Out-Star Model (continued)

- Grossberg's learning is used during the training of out-star model
  w_{ji}(updated) = w_{ji} (previous)+γ (x_i-w_{ji}(previous))
  where i = 1,2,……,m
  s_{jk}(updated) = s_{jk} (previous)+δ (y_k-s_{jk}(previous))
  where k = 1,2,……,n
  x_i^* = w_{ji} (updated)
  y_k^* = s_{jk} (updated)
- It completes one iteration of out-star training.

Now, let us see, how to carry out this particular training of the out-star model and as I told that we are going to use the Grossbergs learning rule for updating of the connecting weights, that is, your w and s. Now, according to your this Grossberg learning rule, $w_{ji}(updated) = w_{ji}(previous) + \gamma(x_i - w_{ji}(previous))$ and here i varies from 1 to up to m, then $s_{jk}(updated) = s_{jk}(previous) + \delta(y_k - s_{jk}(previous))$.

So, by following this actually, we can update the connecting weights and once you have updated then at the end, we can take the decision that $x_i^* = w_{ji}(updated)$ and $y_k^* = s_{jk}(updated)$ and it completes actually one iteration of this out-star training and once completed your this in-star training and the out-star training, now your are in a position to find out, what should be the outputs for a set of inputs.

(Refer Slide Time: 16:07)



Forward-Only CPNN

- It consists of one input layer and one output layer with a common hidden layer, which performs the clustering

Now, if you see like how to implement this actually, we are going to solve one numerical example, but before that, let me try to concentrate on another possibility of this CPNN, that is, your the forward only CPNN. Now, till now, we have considered, we have discussed the full CPNN, where we consider that there are two input layers, two output layers and one hidden layer. Now, for this forward only CPNN, it is simpler in fact, now here, we consider only 1 output layer, only 1 input layer. So, we consider, in fact, your only 1 input layer and only 1 output layer and in between, we have got say 1 hidden layer.

Now, if you see the construction of this forward only CPNN, it looks like this, say we have got 1 input layer consisting of small m number of neurons like x_1, x_i up to x_m, so, this is nothing but the input layer and we have got small n number of your the neurons on the output layer and which indicates actually y_1, y_k and y_n and this is your the hidden layer having p number of neurons and in between your this input layer and the hidden layer, so we have got the in-star model and in between the hidden and the output we have got this particular your out-star model.

The working principle is exactly the same. So, what you will have to do is, you will have to pass the set of inputs and this connecting weights like your u and v will have to go on updating and what you can do is, we can use the Kohonen network, that is unsupervised training sort of thing just to find out, who could be the winner and after that, we can take the help of your the Grossberg learning and once again, we can use both supervised as well as unsupervised learning just to determine like, what should be the set of outputs corresponding to your these particular inputs.

Now, as we have already mentioned that this particular CPNN can be used to model input-output relationships of any engineering process. Now, here, if we compare the performance of this particular CPNN with back propagation neural network or multilayered feed forward network or radial basis function network, there is a possibility that we may not get so much accuracy in this particular CPNN.

But, in CPNN actually there is one advantage, as we do not use the back propagation algorithm or the gradient-based algorithm, so the chance of the solutions for getting trapped into the local minima is nil. So, there is no such chance of local minima problem and that is why, actually many people prefer this particular CPNN, even compared to multilayered feed-forward network.

Thank you.