CSE-8

UNIT-2 N.N DIVYA BHARDWAJ

(NEURAL NETWORKS AND FUZZY LOGIC)

Feedback Networks

Syllabus: Hopfield Nets: structure, training and applications Stability. Back propagation: Concept, Applications and Back Propagation Training Algorithms (Feed Forward Network) Counter Propagation Networks: Kohonan Network, Grossberg Layer and Training, application of counter propagation, Image classification. → (Competitive Network) 36

(A) HOPFIELD NETWORKS:

→ Hopfield net was invented by Dr John J. Hopfield in 1982.

→ It is a kind of feedback network (recurrent) in which the information is returned back from output to the input, thereby giving rise to an iteration process.

→ It is commonly used for auto-association and optimization tasks, pattern recognition problems.

→ It is fully connected, single layer auto associative network means it has only one layer, with each neuron connected to every other neuron.

→ All the neurons act as input and output.

→ Hopfield Networks uses unsupervised learning.

→ There are of two types:-

- Discrete Hopfield net.
- Continuous Hopfield net.

⇒ DISCRETE HOPFIELD NET :-

- Hopfield while working on the magnetic behaviour of the solids (spin glasses) described property of magnetic atoms using two states (+1 & -1). The magnetic mutual exchange between the atoms is represented by a mathematical formula, which led to the development of Hopfield net.

- [The topology of Hopfield network is very simple: it has 'n' neurons which are all networked with each other.]

- [A Hopfield net is able to recognise the unclear pictures correctly. However only one picture can be stored at a time.] Practically, one must assume that many pictures will be given which have to be stored and then classify!

- [The discrete Hopfield net is a fully interconnected neural net with each unit connected to every other unit.]

- [The net has symmetric weights with no self-connections i.e

$$w_{ij} = w_{ji}$$

and $w_{ii} = 0$ (diagonal elements are 0)].

- [Hopfield net differs from iterative auto associative net in the following manner:

Only one unit update its activation at a time

i) Each unit continues to receive an external signal in addition to the signal from the other units in the net.

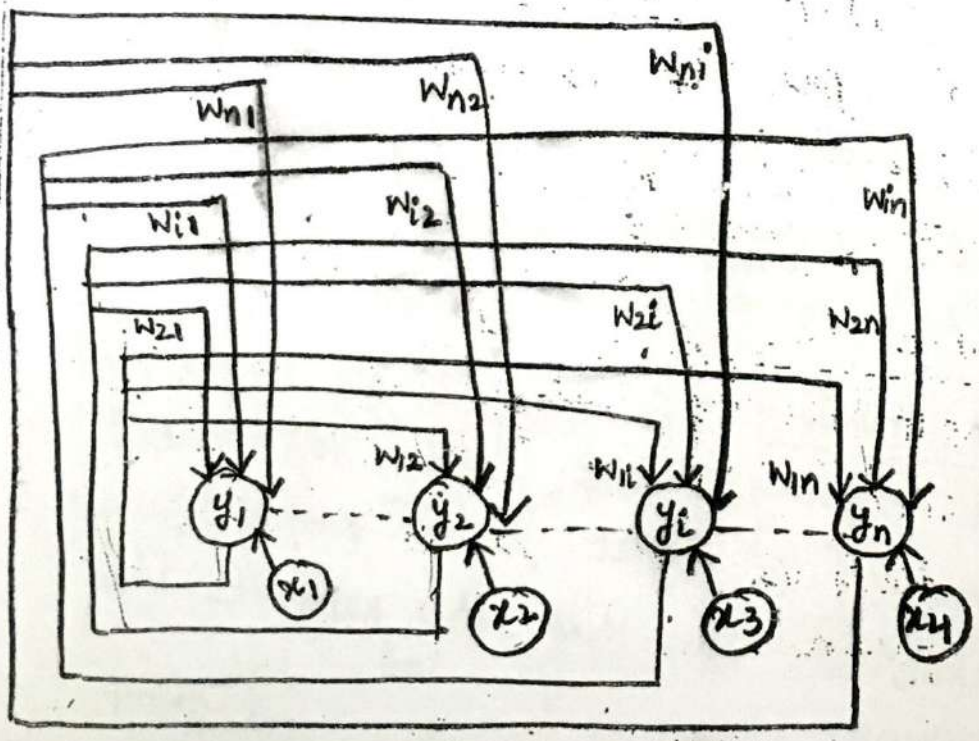
[Brief description about associate memory network :

- These kinds of neural network work on the basis of pattern association, which means they can store different patterns and at the time of giving an output they can produce one of the stored patterns by matching them with the given input pattern.
- They are also called Content - Addressable Memory
- There are two types of associate memories :
  - → Auto associative memory
  - → Hetero associative memory.
- The auto associative memory is a single layer NN in which the input training vector and the output target vector are same. ('n' input, 'n' output). The weights are determined so that the network stores a set of patterns.
- The hetero associative memory is a single layer NN in which the input training vector and the output target vector are not same. ('n' input, 'm' output)]

Continuation with discrete Hopfield net ....

- The asynchronous discrete time updating of the units allows a function known as energy function or Lyapunov function to be found for the net.
- This function proves that the net will converge to a stable set of activations.
- The formulation of the discrete Hopfield net shows the usefulness of the net as a content addressable memory.

(B) Architecture of Hopfield Networks :-



(Architecture of Hopfield Network)

The architecture consist of 'n' no of x input neurons and y output neurons.

- Apart from receiving a signal from input, every neuron is receiving signal from other output neurons also. Thus, there exist a feedback output being returned to each output neuron. Hence, Hopfield network is called a feedback network.

(C) Storage Capacity of Hopfield Network :

- In the Hopfield net, the no of binary patterns that can be stored and recalled in a network with reasonable accuracy is given by,

$$P = 0.15n$$

[P = no of patterns
n = no of neurons]

- If bipolar patterns are used, $P = \frac{n}{2\log_2 n}$

(D) Energy Function of Discrete Hopfield Network :

- Energy function is a parameter that is used to find out the convergence of the network.

It is a non-increasing function that depicts the state of the system.

- Energy function for the discrete Hopfield network is given by:

$$E = -0.5 \sum_{i \neq j} \sum_{j} y_i y_j w_{ij} - \sum x_i y_i + \sum_{i} \theta_i y_i$$

- Energy function is also known as Lyapunov function.

- If the value of energy function is non-increasing then the networks moves towards stability.

(E) ALGORITHM FOR TRAINING DISCRETE HOPFIELD NETWORKS :-

(V. Imp) **

→ Discrete Hopfield Network is described for both binary as well as bipolar vector patterns.

→ Compute Net input (using interconnected weights), apply activation function, then output will be calculated

→ (Imp point) ** It is a kind of unsupervised learning as while computing the output there is no comparison with the target, hence no. of iterations are performed so as to obtain convergence

algorithm of discrete Hopfield net is given as:

Step-I: Initialise weights to store pattern [use Hebb Rule]

$$W = X^T \cdot X$$ ← [Hebbian Rule to obtain weight matrix]

* Convert binary to bipolar for computing weight matrix.

Step-II: for each input vector x, repeat steps 3 to 7

Step-III: Set initial activations of the net (y_i) equal to the external input vector x,

$$y_i = x_i \quad (i = 1, 2 \text{----} n)$$

Step-IV: Perform step 5 to 7 for each unit y_i

Step-V: Compute Net input.

$$y_{ini} = x_i + \sum_j y_j \cdot w_{ji}$$

Step-VI: Determine output using activation function

$$y_i = \begin{cases} 1, & \text{if } y_{ini} > \theta \\ y_i, & \text{if } y_{ini} = \theta \\ 0, & \text{if } y_{ini} < \theta \end{cases}$$

Step-VII: Broadcast the value of y_i to all other units.

Step-VIII: Test for convergence. (No. of iterations are performed)

→ Value of θ is usually taken Zero

→ Order of update of unit is random but each unit has to updated.

Solve following Questions:

Q-1: Consider 3 vectors $S_1 = [1 \ -1 \ 1 \ -1]$

$S_2 = [-1 \ 1 \ 1 \ -1]$

$S_3 = [1 \ 1 \ -1 \ -1]$

Find weight matrix and energy function for 3 input samples. Determine pattern to which $S = [-1 \ 1 \ -1 \ -1]$ associates.

Q-2: Consider a vector $(1 \ 1 \ -1 \ 1)$ to be stored in a net. Test a discrete Hopfield net with mistakes in first component as $(-1 \ 1 \ -1 \ 1)$ of the stored vector.

CONTINUOUS HOPFIELD NETWORKS

It is modified form of discrete Hopfield net.

→ It uses continuous valued output functions, which can be used for associative memory problems or constrained optimization problems.

→ Energy function is:

$$E = -0.5 \sum_{i=1}^{m} \sum_{j=1}^{m}{}' w_{ij} v_i v_j - \sum \theta_i v_i + \frac{1}{\tau} \sum_{i=1}^{m} \int_0^{v_i} f_i^{-1}(v)\,dv$$

τ = time constant

→ Continuous Hopfield Networks are basically used for solving constrained optimization problem such as traveling salesman problem.

(V.Imp)

Difference between Discrete Hopfield net and continuous Hopfield net (CHN)

→ DHN takes only discrete value while CHN takes continuous values.

→ The energy function used is also different.

(G) Applications of Hopfield Networks :

→ Hopfield networks are a form of associative memory initially trained to store a number of patterns & and then it is able to recognise any of the learned patterns by exposure to even corrupted information [Pattern Recognition].

→ Image detection and Recognition.

→ Enhancement of X-Ray images.

→ Medical - Image Restoration.

(II) V Imp

(A) BACK PROPAGATION NETWORKS : (BPN)

Developed by G E Hinton, Rumelhart, R.O Williams in 1986

→ Back propagation is a systematic method for training Multi layer artificial Neural network.

→ It is a feed forward network (a forward flow of information is present).

→ It is trained by supervised learning method (means comparison of output with target occurs)

- If the output does not matches with target, error rate is calculated using gradient descent based delta learning rule commonly called as

ack propagation rule.

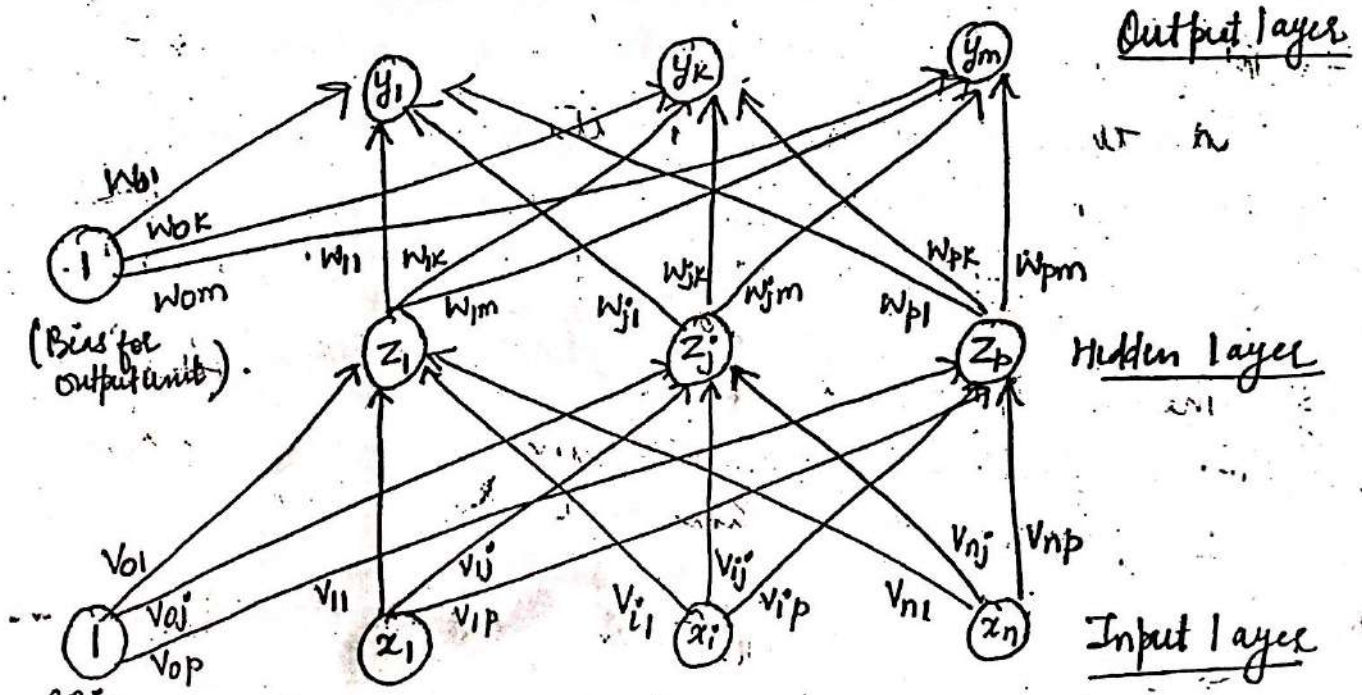
These error rates are moved back into the network from the output layer to hidden layer hence name back propagation.

Architecture of BPN :-

→ BPN Consist of input layers, hidden layers and output layers.

→ Both output unit and hidden (Z) units has bias.

Y output unit has W_{ok} bias, Z hidden unit has V_{oj} as bias



(Architecture of BPN).

→ The increase in the no of layers (hidden layers) results in computational complexity of the network. Hence the time taken for convergence and minimize the error may be very high.

⇒ Training Algorithm of BPN:

The training algo of BPN involves 4 stages.

a) Initialisation of Weights:

During this stage, weights are assigned with some small random values.

b) Feed forward:

→ Each input unit (X_i) receives an input signal and transmits to hidden unit (Z_i).

→ Total net input is calculated at hidden unit and then activation function is applied, the total output calculated at hidden unit is transmitted to layer above i.e Output unit.

→ The output unit again sums up total input received, and activation fn is applied, to produce the response of the network.

(* Keep In mind:- We have calculated 'Net input' twice (once at hidden unit & another at Output unit) Moreover, activation fn is also applied twice) ← During one iteration

Back Propagation of errors :-

e output i.e produced at the Output unit, is compared with target, if it does not matches then error rate is calculated at output unit.

→ The error rate calculated at output unit is propagated backwards to hidden unit. (Hence name BPN)

→ Error rate is also calculated at hidden unit using the error rate of output unit.

(* Keep in mind: error rates at also calculated twice, once at output unit and another at hidden unit).

d) Updation of Weights and Bias :

→ Weights and bias are updated at hidden unit and output unit by using learning rates, and error rates and activations

→ Parameters :

The various parameters used in the training algo are as follows:

x: Input training vector.

$x = (x_1 \text{ ---- } x_i \text{ --- } x_n)$.

t = Output target vector

$t = (t_1 \text{ --- } t_k, \text{ ---- } t_m)$.

δ_k = error at output unit y_k.

δ_j = error at hidden unit z_j.

α = learning rate.

v_{oj} = bias on hidden unit j.

w_{ok} = bias on output unit k.

z_j = hidden unit j.

y_k = output unit k.

→ The training algo used in BPN is as follows.

① Initialization of weights :

Step-I :- Initialize weights to small random values.

Step-II : while stopping condition is false, do steps 3-10.

Step-III :- for each training pair do steps 4-9.

② Feed Forward :

Step-IV :- Each input unit receives the input signal x_i and transmits it to all units in hidden layer.

Step-V : Each hidden unit (z_j ; $j = 1 \cdots p$) sums its weighted input signals

$$z_{in_j} = v_{oj} + \sum_{i=1}^{n} x_i v_{ij}$$

Applying activation fn.

$$z_j = f(z_{inj})$$

and send this signal to output unit.

** Activation fn used in BPN is:

→ (Bipolar sigmoidal or Tangent fn) ⇒ $f(x) = \frac{1-\exp(-\sigma x)}{1+\exp(-\sigma x)}$

→ (Binary sigmoidal or logistic fn) ⇒ $F(x) = \frac{1}{1+\exp(-\sigma x)}$

(σ = steepness parameter).

Step - VI :- Each output unit (y_k, $k = 1 \dots m$) sums its weighted input signal.

$$y_{ink} = w_{0k} + \sum_{j=1}^{p} z_j w_{jk}$$

Applying activation fn to calculate output signals

$$y_k = f(y_{ink})$$

(3) Back Propagation of errors :-

Step - VII :- Each At output unit, the error rates is calculated as:

$$\delta_k = (t_k - y_k) f'(y_{ink})$$

[Use this step only when target is not equal to output]

↑ See here target is used hence supervised learning

$$f'(x) = f(x)\,(1 - f(x))$$ [used this for Calculation of $f'(x)$]

Step - VIII : At hidden unit, sums its delta inputs from output unit

$$\delta_{inj} = \sum_{k=1}^{m} \delta_k w_{jk}$$

Now the error rate at hidden unit is calculated as,

$$\delta_j = \delta_{inj} \cdot f'(z_{inj})$$

(4) Updation of weights and Bias :

Step IX :

At output layer :

$$w_{jk}(new) = w_{jk}(old) + \Delta w_{jk} \quad \leftarrow \text{Weight updation}$$

$$w_{ok}(new) = w_{ok}(old) + \Delta w_{ok} \quad \leftarrow \text{Bias updation}$$

$$\left(\Delta w_{jk} = \alpha \delta_k z_j \;,\; \Delta w_{ok} = \alpha \delta_k\right)$$

At hidden layer :-

$$v_{ij}(new) = v_{ij}(old) + \Delta v_{ij} \quad \leftarrow \text{Weight updation}$$

$$v_{oj}(new) = v_{oj}(old) + \Delta v_{oj} \quad \leftarrow \text{Bias updation}$$

$$\left(\Delta v_{ij} = \alpha \delta_j x_i \;,\; \Delta v_{oj} = \alpha \delta_j\right)$$

Step - X : Test the stopping condition :

The stopping condition may be minimization of the errors, number of epochs.

Selection of Parameters for BPN :-

For efficient operation of BPN it is necessary for the appropriate selection of the parameters used for training.

Initial Weights :-

→ If initial weights is too large, the initial input signal to each hidden or output unit will fall in saturation region where the derivative of the sigmoid has a very small value ($f'(net) = 0$).

→ If initial weights are too small, the net input to a hidden or output unit will approach zero, which causes extremely slow learning.

→ [To get best values of initial weights; (weights and biases) are set to random nos between −0.5 and 0.5 or between −1 and 1].

• Selection of learning Rates :-

3 Methods for adopting learning Rates:

i) Start with high learning rate and steadily dec it.

ii) Inc learning rate in order to improve performance and dec learning rate in order to worsen performance.

iii) Double learning rate until the error value worsens.

⇒ LEARNING IN BACK PROPAGATION :-

V.Imp

There are two types of learning:

i) Sequential learning or pre pattern method :-

A given input pattern is propagated foreward, the error is determined and back propagated, the weights are updated.

ii) Batch learning or pre-epoch pattern :-

The weights are updated only after the entire set of training network has been presented to the network. Thus the weight update is only performed after every epoch.

⇒ In some cases, it is advantageous to accumulate weight correction terms for several patterns and make a single weight adjustment for each weight rather than updating the weights after each pattern is presented. This procedure has a "smoothing effect" Smoothing effect increases the chances of convergence.

→ Momentum factor in BPN :-

V.Imp

- If the momentum is added to the weight update formula, the convergence is faster.
- The weights from one or more previous training patterns must be saved in order to use momentum.

Example : For BPN with momentum, the new weights for training step $t+2$, is based on t and $t+1$.

using momentum, the net doesnot proceed in the direction of the gradient, but travels in the direction of combination of current gradient and the previous direction for which the weight correction is made.

- Main purpose of momentum is to accelerate the convergence of error propagation algorithm.

→ The weight updating formula for BPN with momentum is,

$$W_{jk}(t+1) = W_{jk}(t) + \alpha \delta_k z_j + \mu \left[W_{jk}(t) - W_{jk}(t-1) \right]$$

↑ At the output Layer.

$$V_{jk}(t+1) = V_{jk}(t) + \alpha \delta_j x_i + \mu \left[V_{ij}(t) - V_{ij}(t-1) \right]$$

↑ At the hidden layer

μ is called momentum factor. It ranges from $0 < \mu < 1$.

- Limitations of Momentum factor : (MF)

→ MF can cause weight changes to be in a direction that would increase the error.

→ learning rate places the upper limit on the amount by which the weight can be changed.

→ Merits of BPN :

i) The mathematical formula, present here can be applied to any network and does not require any special mention of the features of the function to be learnt.

ii) Computing time is reduced if the weights chosen are small at the ~~beginning~~ beginning.

iii) The batch update of weights exist, which provides a smoothing effect on weight correction terms.

→ Demerits of BPN :-

i) The no of learning steps may be high, and the learning phase has intensive calculations.

ii) Selection of hidden units in the network is a problem. If no of hidden units is small, then the fn to be learnt may not be possibly represented. If no of hidden units is high then complexity of network may inc.

iii) For complex problem it may require days or weeks to train the network or it may not train at all.

iv) The network may get trapped in a local minima even though there is a much deeper min nearby

v) Training causes temporal instability to system

→ Applications of BPN :-

i) optical character Recognition.
ii) Image Compression
iii) Data compression.
iv) Load forecasting problems in power system area.
v) Control problems
vi) Non linear simulation
vii) fault detection problems.
viii) Face Recognition.

COUNTER PROPAGATION NETWORKS : (CPN)

Concept of CPN:

→ This is a multilayer network based on the various combining structures of input, cluster and output layers.

→ CPN is a combination of 2 algo

- Self organizing map of Kohonen
- Grossberg Outstar

(Together they have properties which are absent in either of the one.

→ CPNs are trained in two stages

Stage-1
Input vectors are clustered on the basis of Euclidean distance or by dot product method.

Stage-2:
The desired response is obtained at the output layer by adjusting the weights from the cluster unit to output unit.

→ CPNs is classified in 2 types

- Full Counter Propagation Network (Full CPN)
- Forward only counter Propagation Network (Forward CPN)

(B) Full CPN :-

→ The Full CPN possess the generalization capability (ability to handle unseen data) which allows it to produce correct output even when the given input is partially incomplete or partially incorrect.

→ Full CPN can represent large no of vector pairs, x:y by constructing a look-up table.

[Full CPN uses a data structure called look-up table to store large no of vector pairs]

→ Full CPN operates in 2 phases:

Phase-1:
The training vectors are used to form clusters. Clusters are formed by using Euclidean distance or by dot product method.

Phase-2:
The weights are adjusted between cluster unit and output unit

[In our numericals, we usually use Euclidean distance for clustering as it is simple to calculate. Moreover, dot product method involves normalization which includes complex calculations].

Architecture of full CPN :-

The given vector pairs are $x : y$ and the approximated vector pairs (output) are $x^* : y^*$

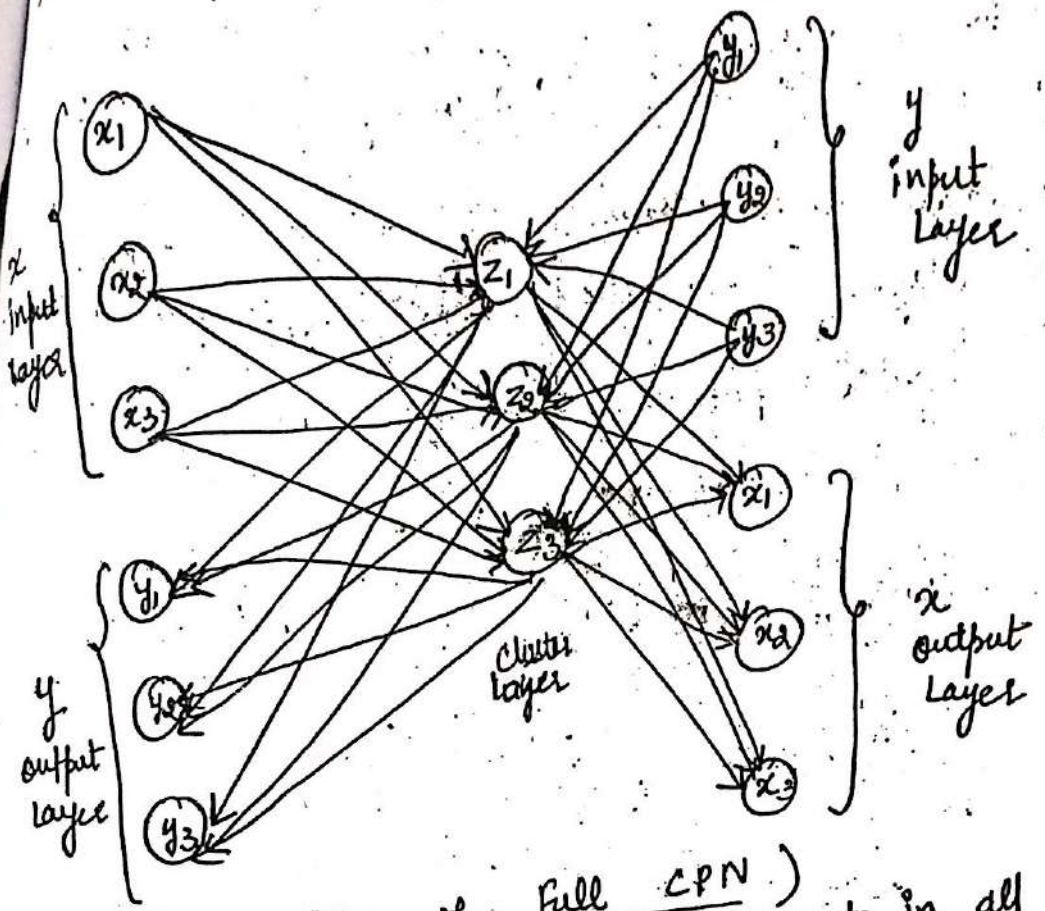


Figure: (Architecture of Full CPN) [3 units in all layers].

→ The architecture consist of 2 input vectors and 2 output vector (x & y).

→ The flow of information is from input to cluster unit and then cluster unit to output unit.

→ The CPN fns in 2 modes:

Normal Mode

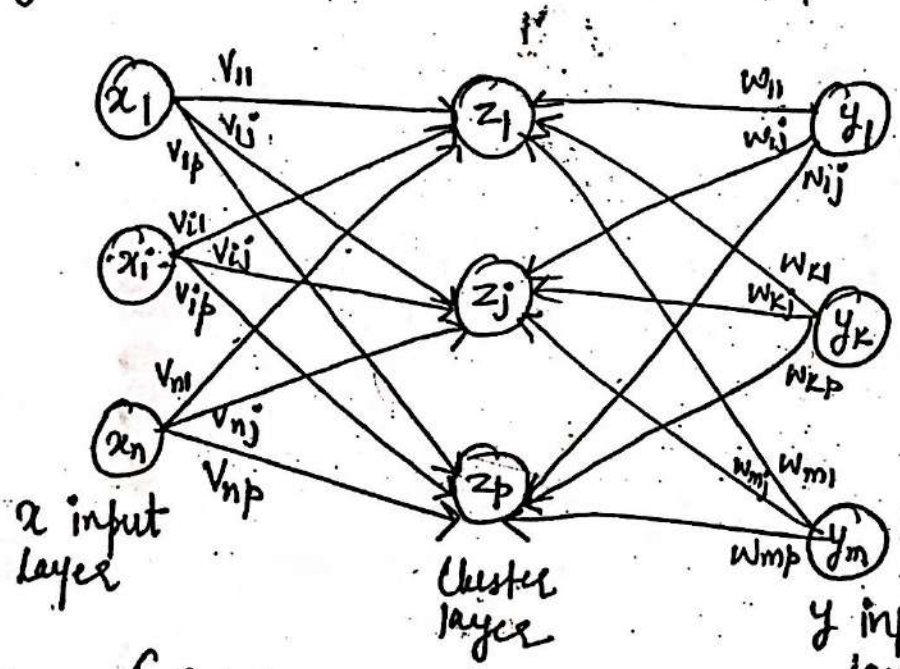
Input vector is accepted and output vector is produced.

Training Mode

Input is applied and Weights are adjusted to obtain desired output vector.

→ The model which connects the input layers to the hidden layers is called instar model.

→ Instar model performs the first phase of training.



(First Phase of full CPN)

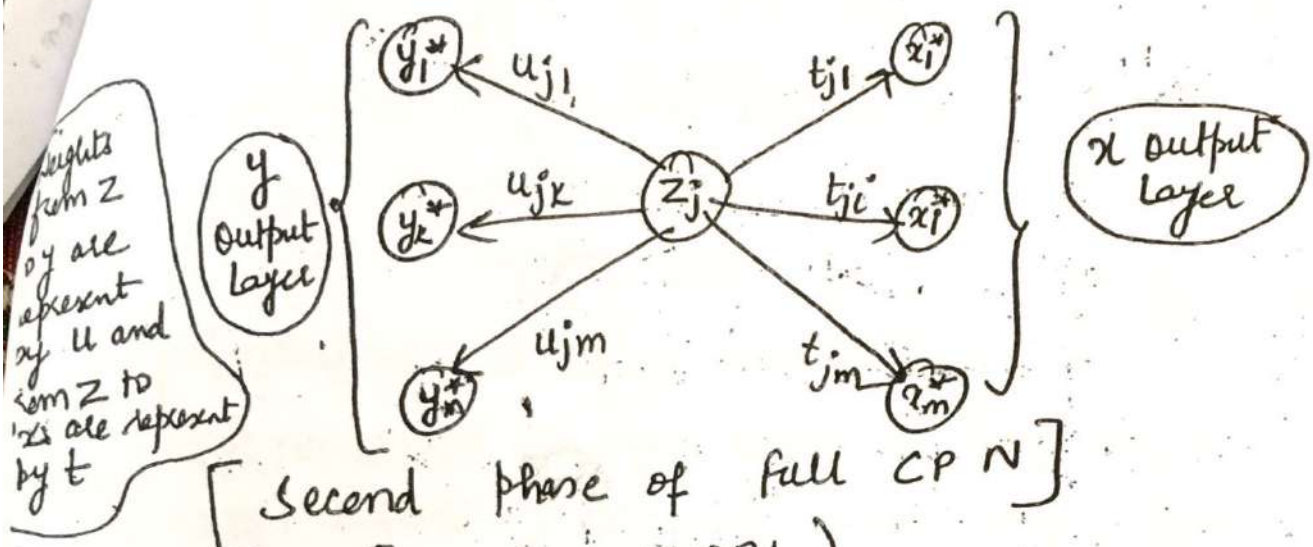
[INSTAR Model]

(All the arrows from the input layers are going inward to cluster layer (hence instar))

from x to Z weights are represented by V

from y to Z weights are represented by W

The model which connects hidden layer to output layer is called Outstar model.



[Second phase of full CPN]

[OUTSTAR MODEL)

(All the arrows are coming out from cluster unit to output unit)

(D) Training Phases of Full CPN :-

First phase
(Instar Modeled Training)

(The winner node uses Kohonen learning rule for its weight updation)

$$v_{ij}^{*}{}_{(new)} = v_{ij}(old) + \alpha\left(x_i - v_{ij}(old)\right)$$

$$w_{kj}(new) = w_{kj}(old) + \beta\left(y_k - w_{jk}(old)\right)$$

Kohonen Learning Rule

Second Phase
[Outstar Modeled Training]

Weights from winner node to output unit are updated using Grossberg rule

Grossberg learning Rule

$$u_{jk}(new) = u_{jk}(old) + \alpha\left(y_k - u_{jk}(old)\right)$$

$$t_{ji}(new) = t_{ji}(old) + b\left(x_i - t_{ji}(old)\right)$$

→ Training Algorithm of FULL CPN:

The parameters used are:

x - Input training vector $x = (x_1 \cdots x_i \cdots x_n)$

y - target output vector $Y = (y_1 \cdots y_k \cdots y_m)$

z_j - activation of cluster unit Z_j.

x^* - approximation of vector x.

y^* - approximation of vector y.

v_{ij} - weight from x input layer to Z-cluster layer.

w_{jk} - weight from y input layer to Z-cluster layer.

t_{ji} - weight from cluster layer to X-output layer.

u_{jk} - weight from cluster layer to Y-output layer.

α, β - learning rates using Kohonen learning

a, b - learning rates using Grossberg learning.

→ The algorithm uses Euclidean distance for calculating winner node.

→ In the phase-1 of training Kohonen learning rule is used.

→ In the phase-2 of training, Grossberg learning rule is used with Kohonen learning rule.

Algorithm for Full CPN is given by:

Phase - I :-

Step-1: Initialize the weights, learning rates etc.

Step-2: while stopping condition for phase I training is false, perform steps 3-8.

Step-3:- For each training input pair x:y do steps 4-6

Step-4: Set X input layer activations to vector x_i;
Set Y input layer activations to vector y_i;

Step-5: Find wining cluster unit using Euclidean distance.

Step-6: Update weight for winning unit.

Kohonen learning

$$v_{ij}(new) = v_{ij}(old) + \alpha\left(x_i - v_{ij}(old)\right); \quad i = 1 \text{ to } n$$

$$w_{kj}(new) = w_{kj}(old) + \beta\left(y_k - w_{kj}(old)\right); \quad k = 1 \text{ to } m$$

Step-7: Reduce learning rates α and β.

Step-8 :- Test stopping condition for phase-I training.

⇒ Phase - II :-

Step-9: while stopping condition for phase-2 training is false, perform steps 10-16.

Step-10: For each training input pair x: y,
do steps 11-14.

Step-11 :- Set X input layer activations to vector x;
Set Y input layer activations to vector y:

Step-12 : find winning cluster unit using Euclidean distance

Step-13 :- Update the weights for winning unit, the value of α and β in this phase are constant

Kohonen learning Rule →

$$v_{ij(new)} = v_{ij(old)} + \alpha(x_i - v_{ij(old)}); \quad i = 1 \text{ to } n$$
$$w_{kj(new)} = w_{kj(old)} + \beta(y_k - w_{kj(old)}); \quad k = 1 \text{ to } m$$

Step-14 : Update weights from unit z_j to output layer

[Grossberg Learning Rule]

$$u_{jk(new)} = u_{jk(old)} + a(y_k - u_{jk(old)}); \quad k = 1 \text{ to } m$$
$$t_{ji(new)} = t_{ji(old)} + b(x_i - t_{ji(old)}); \quad i = 1 \text{ to } n$$

Step-15 : Learning rates a and b are to be reduced.

Step-16 : Test the stopping condition for phase-2 training.

The winning unit selection is done either by dot product or Euclidean distance.

[Stopping condition for CPN may be no of iteration or the reduction in learning rate up to a certain level.

Euclidean distance,

$$D_j = \sum_i (x_i - v_{ij})^2 + \sum_k (y_k - w_{kj})^2$$

The square of whose distance from the input vector is smallest is the winner.

* In case of tie between the selections of the winning unit, the unit with the smallest index is selected'.

keep in mind

See in case of CPN we are not using target hence it is an example of unsupervised learning. Moreover competition exist between the nodes to select winner. Hence CPN is competitive network. (but flow of information is in forward direction)

(E) FORWARD ONLY CPN :-

Concept:

→ It is a simplified form of full CPN.

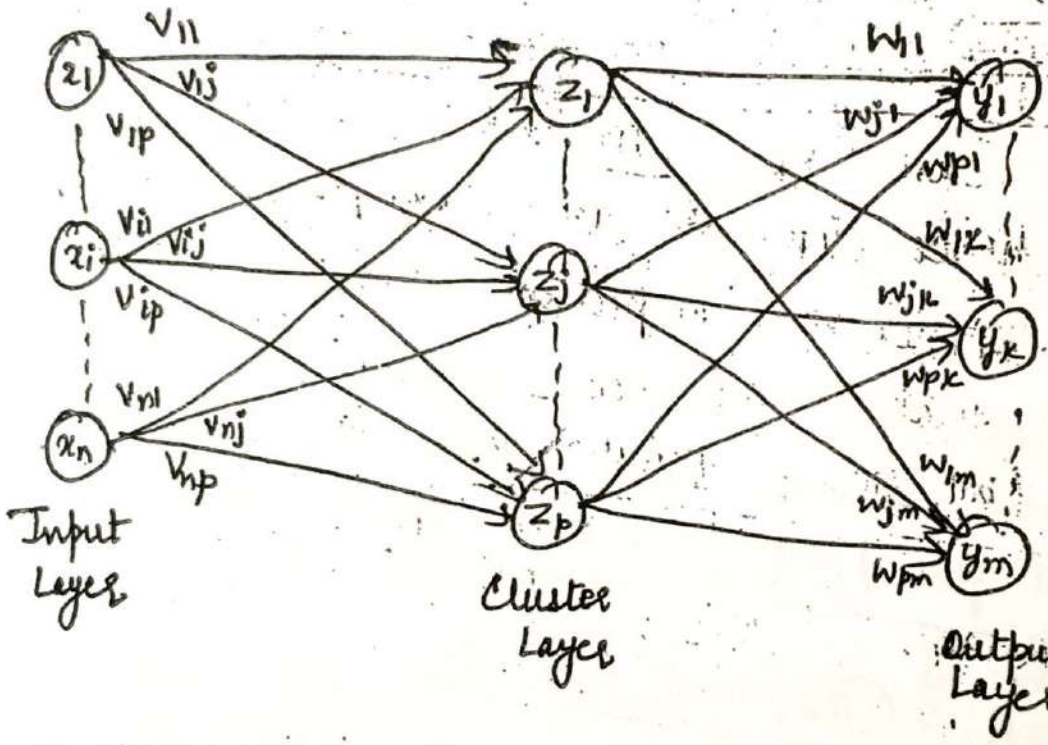
→ It is different from the full-CPN in the sense that it uses only x-vectors to form clusters on the Kohonen units during the first stage of training

→ The forward only CPN has only one input layer, one cluster layer and one output layer and training is performed in 2 phases.

→ Architecture of foreward only CPN :

The architecture of foreward only CPN may look similar to the architecture of Back propagation network but it is not so.

In case of foreward only CPN competition exist between the cluster unit to select winner and only weights of winner node is update.



→ Training Algorithm of foreward only CPN.

(Its similar to full CPN, difference is only one vector is used, and α and a learning rates are used).

training algorithm of forward only CPN

Phase-I

Step-1: Initialize weights, learning rates etc.

Step-2: While stopping condition for phase I is false, perform steps 3-8.

Step-3: For each training input x, perform steps 4-8

Step-4: Set x input layer activations to vector x.

Step-5: Find wining cluster unit.

Step-6: Update weights on wining cluster unit.

Kohonen Rule → $v_{ij}(new) = v_{ij}(old) + \alpha\left(x_i - v_{ij}(old)\right)$; $i = 1$ to n

Step-7: Reduce learning rate α.

Step-8: Test stopping condition for phase-1 training.

→ Phase-II:-

Step-9: While stopping condition is false for phase 2 training do steps: 10-16

Step-10: for each training input pair x:y do steps 11-14

Step-11: Set x input layer activations to vector x;
Set y output layer activations to vector y;

Step-12: Compute winner cluster unit

Step-13: Update weights into unit Z; (cluster unit)

Kohonen Rule → $v_{ij}(new) = v_{ij}(old) + \alpha\left(x_i - v_{ij}(old)\right)$; $i = 1$ to n.

Step-14: Update weights from cluster unit to output unit

Grossberg rule → $w_{jk}(new) = w_{jk}(old) + a\left(y_k - w_{jk}(old)\right); \quad k = 1 \text{ to } m$

Step-15: Reduce learning rate a.

Step-16: Test stopping condition for phase-2 training

Typical value of α and a, may be

$\alpha = 0.5$ to 0.8

$a =$ 0 to 1 (mostly take initial values 0.4 and 0.6)

Winner node is selected by using Euclidean distance

$D(j) = \sum_i (x_i - v_{ij})^2$ is used.

Smaller Dj is winner unit. In case of tie select the node with smaller index

→ Demerits of CPN:

It is inferior to BPN for most mapping network applications.

→ Merits of CPN:-

① It is simple and forms a good statistical model of its input vector environment; ② Network trains rapidly and requires large amount of time.

Applications of CPN:- → Rapid prototyping of systems

→ Capable of generating a fn & its inverse. It has found applications in robotics systems.