PREPARED BY: Dr.S.Kishore

SATHYABAMA UNIVERSITY

SECX1048 FUNDAMENTALS OF FUZZY LOGIC AND ARTIFICIAL NETWORK

UNIT IV DETERMINISTIC & STATISTICAL NETWORKS

Back Propagation Training Algorithm – Practical Difficulties

This was proposed in 1986 by Rumelhart. Back propagation algorithm (BPA) is a way of learning the internal representation of a multilayered network. How back propagation does this is based on a simple idea. The input vector of values pi into the network and get out a corresponding output a3 i. Compare this output to our target (desired) output ti to determine the cumulative error among the different outputs units. But the output units are themselves connected with the hidden units in the network.

One can compute how fast the error changes as we change a hidden activity. But instead of using the desired activities to train the hidden units, the use error derivatives with respect to the hidden activities. Each hidden activity can affect many output units and can therefore have many separate effects on the error. These effects must be combined. The point of the back propagation algorithm is to show how one can compute the error derivatives for all the hidden units efficiently. Once the error derivatives are existing for the hidden activities, its easy to get the error derivatives for the weights going into a hidden unit.

Algorithm: The training involves three stages The feed forward of the input training pattern The back propagation of the associated error The adjustment of the weights Activation function: Back propagation net should have several important characteristics , Continuous , Differentiable ,Monotonically non decreasing One of the most typical activation functions is the binary sigmoid function, which has range of (0,1). Figure 1 is the architectural diagram of BPA.

SECX1048 FUNDAMENTALS OF FUZZY LOGIC AND ARTIFICIAL NETWORK

UNIT 4 DETERMINISTIC & STATISTICAL NETWORKS

PREPARED BY: Dr.S.Kishore

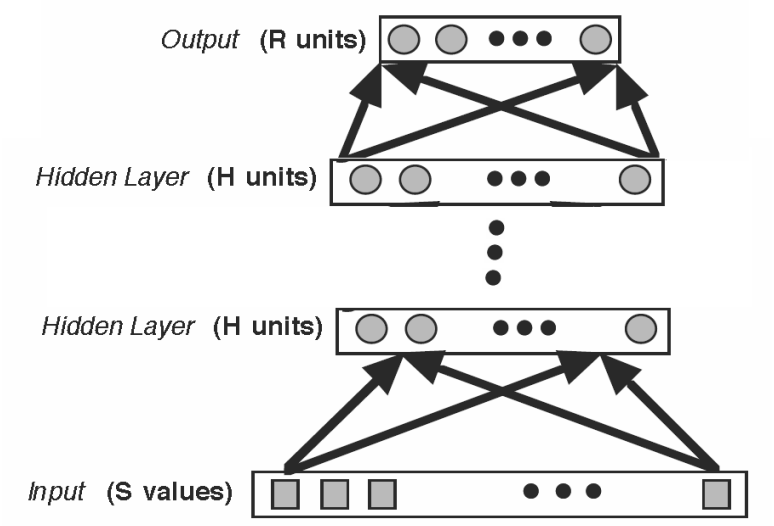


Fig. 1. Architectural diagram

$$f(v_j(n)) = \frac{1}{1+\exp(-av_j(n))} \qquad f'(v_j(n)) = ay_j(n)[1-y_j(n)]$$

Learning Algorithm

$$e_k(n) = d_k(n) - y_k(n)$$

$$\frac{\partial E(n)}{\partial w_{j,k}(n)} = \frac{\partial E(n)}{\partial e\_y_j(n)} \times \frac{\partial e\_y_j(n)}{\partial y_j(n)} \times \frac{\partial y_j(n)}{\partial y\_in_j(n)} \times \frac{\partial y\_in_j(n)}{\partial w_{i,j}(n)}$$

$$\frac{\partial E(n)}{\partial w_{j,k}(n)} = e\_y_k(n) \times -1 \times f'(y\_in_k(n)) \times z_j(n) \qquad \delta_k(n) = -\frac{\partial E(n)}{\partial y\_in_k(n)}$$

Computing $\Delta v_{i,j}$ for none output layers

$$E(n) = \frac{1}{2}\sum_{k\in C} e\_y_k^2(n) \qquad \frac{\partial E(n)}{\partial z_j(n)} = \sum_{k\in C} e\_y_k(n)\frac{\partial e\_y_k(n)}{\partial z_j(n)}$$

$$\frac{\partial e\_y_k(n)}{\partial z_j(n)} = \frac{\partial e\_y_k(n)}{\partial y\_in_k(n)} \times \frac{\partial y\_in_k(n)}{\partial z_j(n)} \qquad e\_y_k(n) = d_k(n) - y_k(n) = d_k(n) - f(y\_in_k)$$

PREPARED BY: Dr.S.Kishore

$$\frac{\partial e\_y_k(n)}{\partial y\_in_k(n)} = -f'(y\_in_k) \qquad \frac{\partial y\_in_k(n)}{\partial z_j(n)} = w_{j,k}(n)$$

Computing $\Delta v_{i,j}$ for none output layers

$$\delta_j(n) = \frac{\partial E(n)}{\partial z_j(n)} \times \frac{\partial z_j(n)}{\partial z\_in_j(n)} = f'(z\_in_j(n)) \times \sum_{k \in C} \delta_k(n) w_{k,j}(n)$$

$$\frac{\partial E(n)}{\partial z_j(n)} = \sum_{k \in C} e\_y_k(n) \frac{\partial e\_y_k(n)}{\partial z_j(n)} \qquad \frac{\partial E(n)}{\partial z_j(n)} = -\sum_{k \in C} e\_y_k(n) f'(y_k(n)) w_{k,j}(n)$$

$$\frac{\partial E(n)}{\partial z_j(n)} = \sum_{k \in C} \delta_k(n) w_{k,j}(n) \qquad z_j(n) = f(z\_in_j(n)) \qquad \frac{\partial z_j(n)}{\partial z\_in_j(n)} = f'(z\_in_j(n))$$

Computing Weight correction

(Weight correction) = (Learning rate parameter) * (local gradient) * (input signal of previouse layer Neuron)

$$\Delta w_{i,j}(n) = \eta \times \delta_j(n) \times x_i(n) \qquad \Delta v_{j,k}(n) = \eta \times \delta_k(n) \times z_j(n)$$

Training Algorithm

Step 0 : Initialize weights (Set to random variables with zero mean and variance one)

Step 1: While stopping condition is false do Step 2.

Step 2: For each training pair do Steps

Step 3: Each input unit(Xi,i=1,..,n) receives input signal xi and broadcasts this signal to all units in the layer above.

Step 4: Each hidden unit (Zj j=1,…,p) sums its weighted input signals applies its activation function to compute its output signal and sends this signal to all units in the layer above (output units).

$$z\_in_j = v_{oj} + \sum_{i=1}^{n} x_i v_{ij} \qquad z_j = f(z\_in_j)$$

Back propagation of error:

Step 6: Each output unit (Yk ,k=1,…,m) receives a target pattern corresponding to the input training pattern computes its error information term.

$$\partial_k = (t_k - y_k) f'(y\_in_k),$$

Calculates its weight correction term (used to update wjk later),

$$\Delta w_{Ok} = \alpha \delta_k Z_j,$$

Calculates its bias correction term (used to update wOk later) and sends to units in the layer below $\Delta w_{Ok} = \alpha \delta_k$

Step 7: Each hidden units (Zj, j=1,…,p) sums its delta inputs from units in the layer above).

O_inj= $\sum_{k=1}^{m} \partial_k w_{jk}$

multiplies by the derivative of its activation function to

calculate its error information term, $\partial_j = \partial\_in_j f'(z\_in_j),$

calculates its weight correction term(used to update vij later), $\Delta v_{ij} = \alpha \partial_j x_i$

and calculates its bias correction term(used to update voj later), $\Delta v_{oj} = \alpha \partial_j$

Update weights and biases:

Step 8: Each output units(Yk,k=1,….,m) updates its bias and weights(j=0,…,p):

wjk(new)=wjk(old)+ Δw_{jk}

Each hidden unit(Zj j=1,….,p) updates its bias and weights(i=0,….,n):

vij(new)=vij(old)+ Δv_{ij}

Step 9: Test stopping condition

Virtues and Limitations of BPA:

- Connectionism
  - Biological Reasons
    - No excitatory or inhibitory for real neurons
    - No Global connection in MLP
    - No backward propagation in real neurons

- Useful in parallel hardware implementation
- Fault Tolerance

- Computational Efficiency
  - Computational complexity of alg. is measured in terms of multiplications,additions…
  - Learning Algorithm is said to be computationally efficient , when its complexity is polynomial…
  - The BP algorithm is computationally efficient.
    - In MLP with a total W weights , its complexity is linear in W
- Convergence
- Saarinen (1992) : Local convergence rates of the BP algorithm are linear

  Too flat OR too curved

  Wrong Direction
- Local Minima

  BP learning is basically a Hill climbing technique.

  Presence of local minima (isolated valleys)

Counter propagation network: Structure & Operation – Training of Kohonen and Grossberg Layer

They are multilayer networks based on a combination of input, clustering and output layers. Counter propagation networks are trained in two stages: First stage; the input vectors are clustered. No topology was assumed for the cluster units. Second stage; the weights from the cluster units to the output units are adapted to produce the desired response.

There are two types of counter propagation nets: Full counter propagation. Forward only counter propagation.

Full counter propagation: This was developed to provide an efficient method of representing a large number of vector pairs, x:y by adaptively constructing a look-up table. It produces an approximation: based on input of an x vector or input of a y vector

only, or input of an x:y pair, possibly with some distorted or missing elements in either or both vectors. Produces an approximation x*:y* based on input of an x vector, input of a y vector only and input of an x:y ,possibly with some distorted or missing elements in either or both vectors. Figure 2 depicts the training process. This takes place in two phases.

Phase 1: The units in the cluster layer compete. The learning rule for weight updates on the winning cluster unit is (only the winning unit is allowed to learn)

$$\begin{cases} w_{iJ}^{new} = w_{iJ}^{old} + \alpha\left(x_i - w_{iJ}^{old}\right) & i = 1,2,...,n \\ u_{kJ}^{new} = w_{kJ}^{old} + \beta\left(y_k - u_{kJ}^{old}\right) & k = 1,2,...,m \end{cases}$$

(This is standard Kohonen learning)

Phase 2: The weights from the winning cluster unit J to the output units are adjusted so that the vector of activations of the units in the Y output layer, y*, is an approximation to the input vector y; x*, is an approximation to the input vector x. The weight updates for the units in the Y output and X output layers are,

$$\begin{cases} v_{Jk}^{new} = v_{Jk}^{old} + a\left(y_k - v_{Jk}^{old}\right) & k = 1,2,...,m \\ t_{Ji}^{new} = t_{Ji}^{old} + b\left(x_i - t_{Ji}^{old}\right) & i = 1,2,...,n \end{cases}$$

(This is known as Grossberg learning)

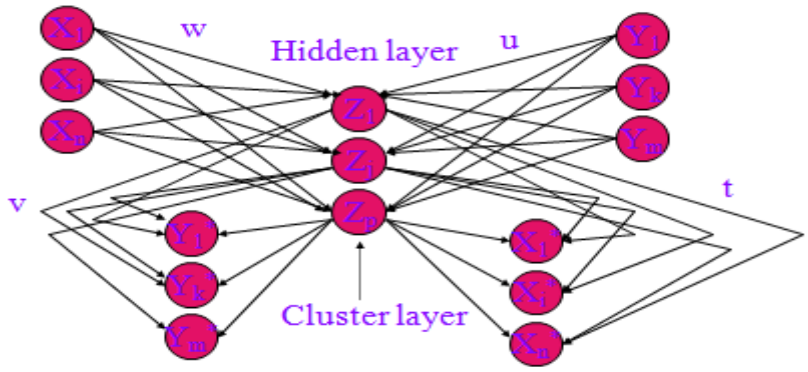


Fig.2 Training

SECX1048 FUNDAMENTALS OF FUZZY LOGIC AND ARTIFICIAL NETWORK

UNIT 4 DETERMINISTIC & STATISTICAL NETWORKS

PREPARED BY: Dr.S.Kishore

x : input training vector : $x = (x_1,...,x_i,...,x_n)$

y : target output corresponding to input x : $y = (y_1,...,y_k,...,y_m)$

z_j : activation of cluster layer unit Z_j

x^* : computed approximation to vector x

y^* : computed approximation to vector y

w_{ij} : weight from X input layer, unit X_i, to cluster layer, unit Z_j

u_{kj} : weight from Y input layer, unit Y_k, to cluster layer, unit Z_j

v_{jk} : weight from cluster layer, unit Z_j, to Y output layer, unit Y_k^*

t_{jk} : weight from cluster layer, unit Z_j, to X output layer, unit X_i^*

α, β : learning rates for weights into cluster layer (Kohonen learning)

a, b : learning rates for weight out from cluster layer (Grossberg learning)

Phase 1:

Step 1. Initialize weights, learning rates, etc.

Step 2. While stopping condition for Phase 1 is false, do Step 3-8

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

Step 4. Set *X* input layer activations to vector x ; set *Y* input layer activations to vector y.

Step 5. Find winning cluster unit; call its index *J*

Step 6. Update weights for unit *ZJ*

Step 7. Reduce learning rate α and β.

Step 8. Test stopping condition for Phase 1 training

Phase 2:

Step 9. While stopping condition for Phase 2 is false, do Step 10-16

Step 10. For each training input pair x:y, do Step 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; call its index *J*

Step 13. Update weights for unit *ZJ*:

Step 14. Update weights from unit *ZJ* to the output layers

Step 15. Reduce learning rate *a* and *b*.

Step 16. Test stopping condition for Phase 2 training.

Applications of BPN & CPN

The application for counter propagation is as follows:

Step0: initialize weights.

step1: for each input pair x:y, do step 2-4.

Step2: set X input layer activation to vector x

set Y input layer activation to vector Y;

Step3: find cluster unit Z, that is closest to the input pair

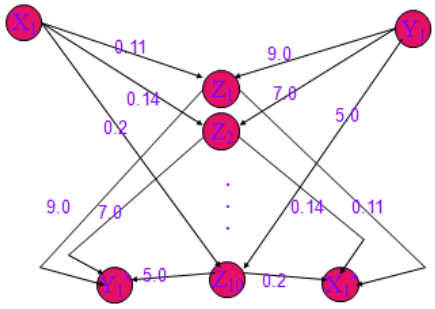
Step4: compute approximations to x and y:

$X^*_i=t_{ji}$

$Y^*_k=u_{jk}$

Function approximation of y=1/x, after training phase we have:

Cluster unit	v	w
z1	0.11	9.0
z2	0.14	7.0
z3	0.20	5.0
z4	0.30	3.3
z5	0.60	1.6
z6	1.60	0.6
z7	3.30	0.3
z8	5.00	0.2
z9	7.00	0.14
z10	9.00	0.11



To approximate value for y for x=0.12. As we don't know anything about y compute D just by means of x.

D1=(.12-.11)^2 =.0001

D2=.0004

D3=.064

D4=.032

D5=.23

D6=2.2

D7=10.1

D8=23.8

D9=47.3

D10=81

Statistical Method – Training Application – Boltzman Training- Cauchy's training:

System of multi-particles, each particle can change its state. Hard to know the system's exact state/config. , and its energy.

Statistical approach: probability of the system is at a given state. Assume all possible states obey Boltzmann-Gibbs distribution:

The energy when the system is at state α

The probability the system is at state α

$P_\alpha = \frac{1}{z} e^{-\beta E_\alpha}$, where $z = \sum_\alpha e^{-\beta E_\alpha}$ is the normalizat ion factor so $\sum_r P_\alpha = 1$

$\beta = (K_B T)^{-1}$, where K_B : Boltzmann constant, T : absolute temperatu re

Ingnoring K_B and using T for artificial temperatu re

$P_\alpha = \frac{1}{z} e^{-E_\alpha / T}$, and $\frac{P_\alpha}{P_\beta} = \frac{e^{-E_\alpha / T}}{e^{-E_\beta / T}} = e^{-(E_\alpha - E_\beta)/T} = e^{-\Delta E / T}$

Boltzmann Machine Architecture (BM):

A set of visible units: nodes can be accessed from outside

A set of hidden units: Adding hidden nodes to increase the computing power

Connection between nodes:

- Fully connected between any two nodes (not layered)
- Symmetric connection: $w_{ij} = w_{ji}$ $w_{ii} = 0$

Nodes are the same as in discrete HM $net_i = \sum_{j \neq i} w_{ij} x_j + \theta_i$

Energy function: $E = -\frac{1}{2} \sum_i \sum_{j \neq i} w_{ij} x_i x_j - \sum_i \theta_i x_i$

BM computing (SA), with a given set of weights:

1. Apply an input pattern to the visible units.

Some components may be missing---pattern completion;

Some components may be permanently clamped to the input values (as recall key or problem input parameters).

2. Assign randomly 0/1 to all unknown units

3. Perform SA process according to a given cooling schedule. Specifically, at any given temperature T. An random picked non-clamped unit i is assigned value of 1 with probability $(1 + e^{-net_i / T})^{-1}$ and 0 with probability $1 - (1 + e^{-net_i / T})^{-1}$

- BM learning (obtaining weights from examples)
  - Probability distribution of visible vectors in the environment.
  - Examples: randomly drawn from the entire population of possible visible vectors.

- Construct a model of the environment that has the same probability distribution of visible nodes as the one in the example set.

There may be many models satisfying this condition

- Because the model involves hidden units.

BM Learning rule:

$\{V_a\}$: The set of example (visible vectors)

$\{H_b\}$: The set of vectors appearing on the hidden units

Two phases:

- Clamping phase: each example is clamped to visible units.
- free-run phase: none of the visible unit is clamped

: Probability that example is applied in Clamping phase

: Probability that the system is stabilized with at visible units in free-run

$$\frac{\partial G}{\partial w_{ij}} = \sum_a P^+(V_a) \frac{\partial}{\partial w_{ij}} \ln \frac{P^+(V_a)}{P^-(V_a)}$$

$$= \sum_a P^+(V_a) \frac{P^-(V_a)}{P^+(V_a)} \frac{\partial}{\partial w_{ij}} \frac{P^+(V_a)}{P^-(V_a)}$$

$$= \sum_a P^-(V_a) \frac{-P^+(V_a)}{\left(P^-(V_a)\right)^2} \frac{\partial}{\partial w_{ij}} P^-(V_a)$$

$$= -\sum_a \frac{P^+(V_a)}{P^-(V_a)} \frac{\partial}{\partial w_{ij}} P^-(V_a)$$

BM Learning algorithm

1. Compute P^+_{ij}

1.1. Clamp one training vector to the visible units of the network

1.2. Anneal the network according to the annealing schedule until equilibrium is reached at the desired minimum temperature.

1.3. Continue to run the network for several more cycles.

After each cycle, determine P^{-}_{ij} which pairs of connected unit are "on" simultaneously.

1.4. Average the co-occurrence results from 1.3

1.5. Repeat steps 1.1 to 1.4 for all training vectors and average the co-occurrence results to estimate$P^{+}_{ij}$for each pair of connected units.

2. Compute the same steps as 1.1 to 1.5 except no visible unit is clamped.

3. Calculate and apply weight change $\Delta w_{ij} = \mu(p_{ij}^{+} - p_{ij}^{-})$

4. Repeat steps 1 to 3 until $p_{ij}^{+} - p_{ij}$is sufficiently small.

Comments:

1. BM is a stochastic machines not a deterministic one.
2. It has higher representative/computation power than HM+SA (due to the existence of hidden nodes).
3. Since learning takes gradient descent approach, only local optimal result is guaranteed (computation still guarantees global optimal if temperature decreases infinitely slow during SA).
4. Learning can be extremely slow, due to repeated SA involved
5. Speed up:

   Hardware implementation

   Mean field theory: turning BM to deterministic.

$$\text{approximating } P_{ij} \text{ by } \langle x_i \rangle \langle x_j \rangle$$

$$\langle x_i \rangle = \tanh\left[\frac{1}{T}\left(\sum w_{ij} \langle x_j \rangle\right)\right]$$

Hop Field Network and Boltzman Machine:

Hopfield Network: The net is a fully interconnected neural net, in the sense that each unit is connected to every other unit. The net has symmetric weights with no self-connections, Wij =Wji And Wii =0.

Only one unit updates its activation at a time and each unit continues to receive an external signal in addition to the signal from the other units in the net. The

asynchronous updating of the units allows a function, known as an energy or Lyapunov function, to be found for the net.

Architecture: An expanded form of a common representation of the Hopfield net is as shown in the figure 3.

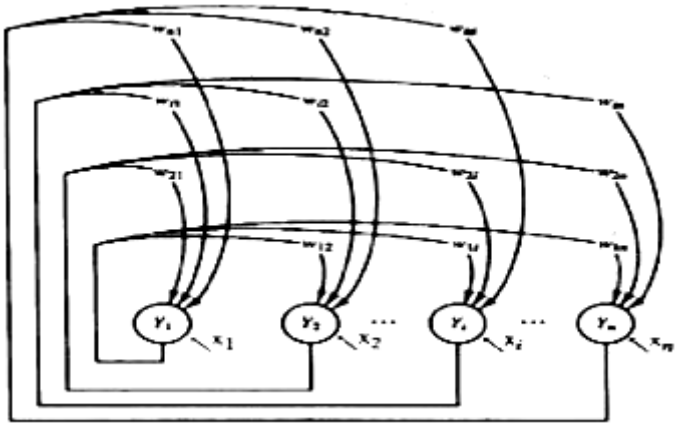


Fig.3. Hopfield Net

Speed Energy Function – Network Capacity

Proposals typically suggest turning the radio off when not needed. Power Saving Mode in IEEE 802.11 (Infrastructure Mode). An Access Point periodically transmits a beacon indicating which nodes have packets waiting for them. Each power saving (PS) node wakes up periodically to receive the beacon. If the capacity were to be equally divided. Now if source and destination pair were 1m away. Throughput and Transport Capacity would be equal. It should be noted that transport capacity increases when the signal power decays more rapidly with distance. Each node randomly chooses destination. Destination chosen independently as the node closest to a randomly located point. All transmissions use the same range. Nodes are randomly located either on the surface of a sphere or in a plane.

Applications of A/D network:

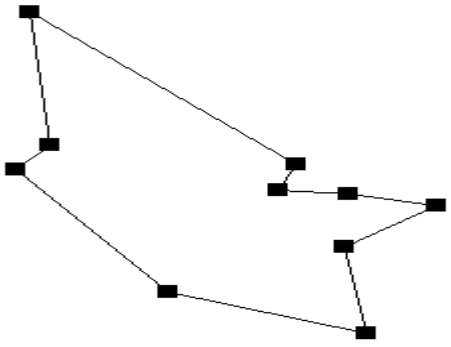
A recurrent network is used to produce a four bit A/D converter. Artificial neurons serve as amplifier weights representing resistors and convert each neuron output to input of other circuits. To achieve stability resistors are not connected between a neuron output

and its occurrence as input. As weights are to be symmetrical, value of resistors from output of neuron j to i. The training table consists of a continuously increasing voltage x applied to single input terminal which produces 4 output representing a binary number. Hopfield network algorithm is applied and steps are repeated till the output is as close as possible to analog value of input.

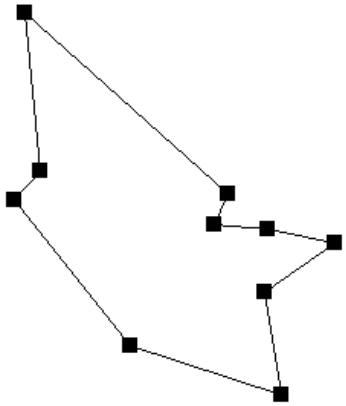
Travelling Sales Man Problem (TSP):

Given a complete, weighted graph on n nodes, find the least weight Hamiltonian cycle, a cycle that visits every node once. Though this problem is easy enough to explain, it is very difficult to solve. Finding all the Hamiltonian cycles of a graph takes exponential time. Therefore, TSP is in the class NP. The TSP is interesting because it is in the class that is called NP-complete. If the TSP is found to have an algorithm that does not take exponential time, the other problems that are NP-complete will also be solved, and vice-a-versa.

The TSP has quite a history. In the Odyssey by Homer, Ulysses has to travel to 16 cities. 653,837,184,000 distinct routes are possible. One of the first TSP papers was published in the 1920s.



Cheapest



Farthest

One of the good attributes of these 2 heuristics is that they avoid the possibility of edge- crossing. The crossing of edges guarantees that the solution is not optimal. There exists an algorithm that removes all the edge-crossings in at most n^3 time. While good solutions may be obtained using heuristics, it is difficult to prove if those solutions are optimal. Perhaps there is a way that is smarter than brute force that gives the optimal solution.

Applications of the TSP:

Routing around Cities
Computer Wiring - connecting together computer components using minimum wire length
Archaeological Seriation - ordering sites in time
Genome Sequencing - arranging DNA fragments in sequence
Job Sequencing - sequencing jobs in order to minimise total set-up time between jobs

RBF Network

This is becoming an increasingly popular neural network with diverse applications and is probably the main rival to the multi-layered Perceptron. Much of the inspiration for RBF networks has come from traditional statistical pattern classification techniques. The basic architecture for a RBF is a 3-layer network, as shown in Figure 4.

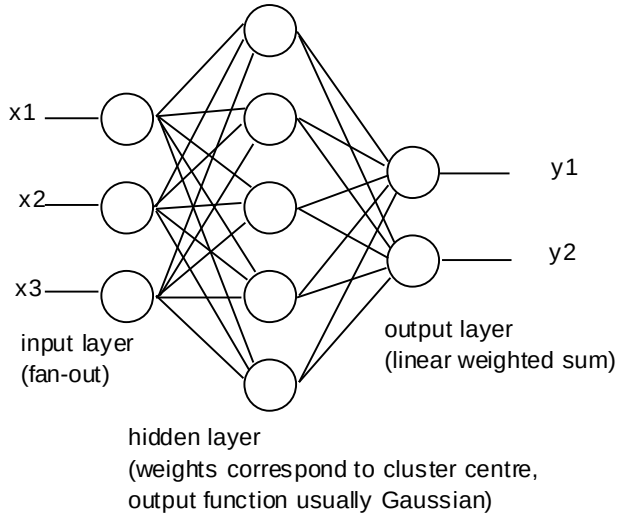


Fig.4. RBF Network

The input layer is simply a fan-out layer and does no processing. The second or hidden layer performs a non-linear mapping from the input space into a (usually) higher dimensional space in which the patterns become linearly separable.

Output layer:

The final layer performs a simple weighted sum with a linear output.

If the RBF network is used for function approximation (matching a real number) then this output is fine. However, if pattern classification is required, then a hard-limiter or sigmoid function could be placed on the output neurons to give 0/1 output values.

Clustering:

The unique feature of the RBF network is the process performed in the hidden layer. The idea is that the patterns in the input space form clusters.

- If the centres of these clusters are known, then the distance from the cluster centre can be measured. Furthermore, this distance measure is made non-linear,

so that if a pattern is in an area that is close to a cluster centre it gives a value close to 1. Beyond this area, the value drops dramatically. The notion is that this area is radially symmetrical around the cluster centre, so that the non-linear function becomes known as the radial-basis function.

Distance Measure:

The distance measured from the cluster centre is usually the Euclidean distance. For each neuron in the hidden layer, the weights represent the co-ordinates of the centre of the cluster. Therefore, when that neuron receives an input pattern, *X*, the distance is found using the following equation: $r_j = \sqrt{\sum_{i=1}^{n}(x_i - w_{ij})^2}$

Width of hidden unit basis function $(hidden\_unit)\varphi_j = \exp(-\frac{\sum_{i=1}^{n}(x_i - w_{ij})^2}{2\sigma^2})$

The variable sigma, σ, defines the width or radius of the bell-shape and is something that has to be determined empirically. When the distance from the centre of the Gaussian reaches σ, the output drops from 1 to 0.6.

Training hidden layer:

The hidden layer in a RBF network has units which have weights that correspond to the vector representation of the centre of a cluster. These weights are found either using a traditional clustering algorithm such as the *k*-means algorithm, or adaptively using essentially the Kohonen algorithm. In either case, the training is unsupervised but the number of clusters that you expect, *k*, is set in advance. The algorithms then find the best fit to these clusters. The *k* -means algorithm will be briefly outlined. Initially *k* points in the pattern space are randomly set. Then for each item of data in the training set, the distances are found from all of the *k* centres.

The closest centre is chosen for each item of data this is the initial classification, so all items of data will be assigned a class from 1 to *k*. Then, for all data which has been found to be class 1, the average or mean values are found for each of co-ordinates.

These become the new values for the centre corresponding to class 1. Repeated for all data found to be in class 2, then class 3 and so on until class *k* is dealt with, now have *k* new centres. Process of measuring the distance between the centres and each item of data and re-classifying the data is repeated until there is no further change i.e. the sum of the distances monitored and training halts when the total distance no longer falls.

Adaptive k-means:

The alternative is to use an adaptive *k* -means algorithm which similar to Kohonen learning. Input patterns are presented to all of the cluster centres one at a time, and the cluster centres adjusted after each one. The cluster centre that is nearest to the input data wins, and is shifted slightly towards the new data. This has the advantage that you don't have to store all of the training data so can be done on-line.

Finding radius of Gaussians:

Having found the cluster centres using one or other of these methods, the next step is determining the radius of the Gaussian curves. This is usually done using the *P*-nearest neighbour algorithm. A number *P* is chosen, and for each centre, the *P* nearest centres is found. The root-mean squared distance between the current cluster centre and its *P* nearest neighbours is calculated, and this is the value chosen for σ. So, if the current cluster centre is c_j, the value is:

$$\sigma_j = \sqrt{\frac{1}{P}\sum_{i=1}^{P}(c_k - c_i)^2}$$

A typical value for *P* is 2, in which case σ is set to be the average distance from the two nearest neighbouring cluster centres. Using this method XOR function can be implemented using a minimum of 4 hidden units. If more than four units are used, the additional units duplicate the centres and therefore do not contribute any further discrimination to the network. So, assuming four neurons in the hidden layer, each unit is centred on one of the four input patterns, namely 00, 01, 10 and 11.

The *P*-nearest neighbour algorithm with *P* set to 2 is used to find the size if the radii. In each of the neurons, the distances to the other three neurons is 1, 1 and 1.414, so the two nearest cluster centres are at a distance of 1. Using the mean squared distance as the radii gives each neuron a radius of 1.

Training output layer: Having trained the hidden layer with some unsupervised learning, the final step is to train the output layer using a standard gradient descent technique such as the Least Mean Squares algorithm. In the example of the exclusive-or function given above a suitable set of weights would be +1, –1, –1 and +1.

Advantages/Disadvantages:

RBF trains faster than a MLP. Another advantage that is claimed is that the hidden layer is easier to interpret than the hidden layer in an MLP. Although the RBF is quick to train, when training is finished and it is being used it is slower than a MLP, so where speed is a factor a MLP may be more appropriate.

SECX1048 FUNDAMENTALS OF FUZZY LOGIC AND ARTIFICIAL NETWORK

UNIT 4 DETERMINISTIC & STATISTICAL NETWORKS

PREPARED BY: Dr.S.Kishore

QUESTIONS FOR PRACTICE

PART A

1. What is a back propagation algorithm?
2. What is training?
3. Define CPN?
4. What are the applications of BP?
5. Brief about grossberg layer?
6. List out statistical methods of training?
7. What is Boltzmann training?
8. Brief about network capacity?

PART B

1. Explain about back propagation algorithm in detail and its limitations?
2. Discuss about counter propagation networks in detail?
3. Elaborate on the training of kohonen and grossberg layers?
4. Discuss about Boltzmann machine, training and limitations?
5. Explain travelling sales man problem in detail?
6. Discuss in detail about RBF networks?