UNIT 7 APPLICATIONS OF MLP

Structure **Page No.**

7.1 INTRODUCTION

In the previous units of neural networks, we discussed the general introduction to neural networks. This involved the model of neuron, the McCulloch-Pits neuron, network topologies, learning methods, as well as properties and applications of neural networks. In this unit, we shall start with function approximation in Sec. 7.2. Also, we shall discuss learning and generalization in Sec. 7.3 and selection of architecture in Sec. 7.4.

Objectives

After studying this unit you should be able to:

- define the function approximation in neural networks;
- apply generalization in the context of neural networks;
- find the selection of architecture for neural networks.

7.2 FUNCTION APPROXIMATION

Most applications of neural networks utilize the function approximation capability. These include associative memory, modeling, signal processing, pattern recognition, regression and prediction and system identification. Image restoration is also an important function approximation problem. For this, the supervised learning is used in the neural networks. The input and output layers needed in the neural networks are defined on the basis of the dimension of the input and output layer patterns. And the networks parameters are adjusted accordingly. A trained network represents the learned non-linear relation between the input-output pairs, and can generalize where an unknown input pattern is presented to the network.

On one hand, the capability of universal function approximation lays the theoretical foundation for the wide application of forward neural networks, on the other hand, recurrent neural networks using the sigmoid activation function are Turing equivalent and can compute whatever function any digital computer.

Some aspects of the representation capability of forward neural networks are described below.

1. Boolean Function Approximation

To represent logic or Boolean functions, the forward neural networks (FNNs) with binary neurons are used binary neural networks, the input and output values for each neuron are represented either by Boolean variables (0 or 1) or by bipolar (– 1 or +1). For N independent Boolean variables, there are 2^N combinations of these variables.

This leads to a total of 2^{2^N} different Boolean functions of N variables. Two class can always be discriminated by a **linear threshold gate (LTG)**. Now using a network of LTG, any Boolean function becomes realizable.

The number of linearly separable dichotomies of k points in general position in R^n is found using the function counting theorem which essentially estimates the separating capability of an LTG. The function counting theorem is given below.

Theorem 1 (Function counting theorem): The number of linearly separable dichotomies of m points in general position in R^n is

$$C(k,n)=\begin{cases}2\sum_{i=0}^{n}C(k-1,i), & k>n+1\\ 2^k, & k\le n+1\end{cases} \tag{1}$$

A set of k points in R^n is said to be in a general position if every subset of n or fewer points is linearly independent.

The total number of possible dichotomies of k points is 2^k. Under the assumption of 2^k equiprobable dichotomies, the probability of a single LTG with n inputs to separate k points in general position is given by

$$P(k,n)=\frac{C(k,n)}{2^k}=\begin{cases}2^{1-k}\sum_{i=0}^{n}C(k-1,i), & k>n+1\\ 1, & k\le n+1\end{cases} \tag{2}$$

The fraction P(k,n) is said to be the probability of linear dichotomy. This relation is visualizes in Fig 1.

Here using Eqn. (2), we get

$$P=1 \qquad \text{if } \frac{k}{n+1}\le 1$$

$$P\to 1, n\to\infty \quad \text{if } 1<\frac{k}{n+1}<2$$

$$P=1/2 \qquad \text{if } \frac{k}{n+1}=2$$

Usually, $\frac{k}{n+1}=2$ is used to characterize the statistical capability of a single LTG.

A three-layer $\left(N-2^N-1\right)$ feed forward LTG network can represent any Boolean function with N arguments. To realize an arbitrary function $f:R^N\to\{0,1\}$ defined on M arbitrary points in R^N, the lower bound for the number of hidden nodes is derived as $O\left(\frac{M}{N\log_2\frac{M}{N}}\right)$ for $M\ge 3N$ and $N\to\infty$; for M points in general position, the lower bound is $\frac{M}{2N}$ when $N\to\infty$. Networks with two or more hidden layers are found to be potentially more size efficient than networks with a single hidden layer.

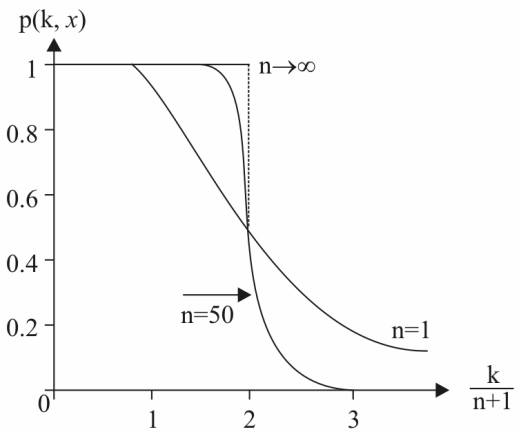


Fig. 1: The probability of linear dichotomy of k points in n dimensions

2. Linear Separability and Nonlinear Separability

Consider a set **X** and M pattern x_i of N dimensions, such that each belonging to one of two classes C_1 and C_2, then such a classification problem is said to be linearly separable. A single LTG can realize linearly separable dichotomy function, characterized by a linear separating surface (hyper plane)

$$\mathbf{w}^T\mathbf{x} + w_0 = 0 \tag{3}$$

where,

w is an N-dimensional vector, and w_0 is a bias toward the origin.

For a pattern,

if $\mathbf{w}^T\mathbf{x} + w_0 > 0$, the pattern belongs to the class C_1

and if $\mathbf{w}^T\mathbf{x} + w_0 < 0$, the pattern belongs to the class C_2.

Some examples of linearly separable classes and linearly inseparable classes in two-dimensional space are shown in Fig. 2, where dots and circles denote patterns of different classes. Here, it can also be noted that these two classes are linearly separable with $x_1 - x_2 = 0$ as the delimiter.

A linearly inseparable dichotomy can become nonlinearly separable. A dichotomy $\{C_1, C_2\}$ of set X is said to be f-separable if there exists a mapping $f : R^N \to R^{N'}$ that satisfies a separating surface

$$\mathbf{w}^T f(\mathbf{x}) = 0 \tag{4}$$

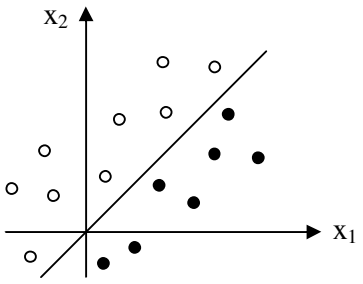


Fig. 2: Linearly separable classification in 2-D

Such that $\mathbf{w}^T f(\mathbf{x}) > 0$ if $\mathbf{x} \in C_1$ and $\mathbf{w}^T f(\mathbf{x}) < 0$ if $\mathbf{x} \in C_2$. Here **w** is a N'-dimensional

vector. As shown in Fig. 3, the two linearly inseparable dichotomies become f-separable. Also, Fig. 3 (a) is the XOR problem. It can be noted here that linearly inseparable classification in cases (a) and (b) become non-linearly separable where the separation surfaces are ellipses.

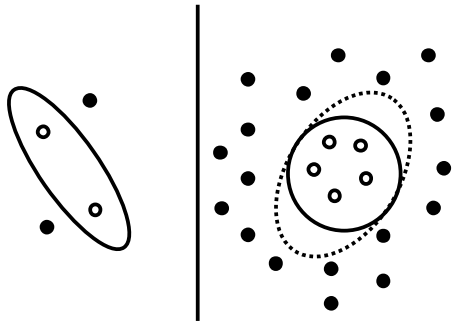


Fig. 3 (a and b): Linear inseparable classification in 2D

The nonlinearly separable problem can be realized by using a polynomial threshold gate (PTG), which changes the linear term in the LTG into high-order polynomials. The function counting theorem is also applicable to PTGs.

The function counting theorem still holds true if the set of k points is in general position in f-space, that is, the set of k points is in f-general position.

3. Continuous Function Approximation

A three-layer forward neural network with a sufficient number of hidden units can approximate any continuous function to any degree of accuracy. This is guaranteed by Kolmogorov's theorem, which is stated below:

Theorem 2 (Kolmogorov): Any continuous real-valued function $F(x_1,\cdots,x_n)$ defined on $[0,1]^n$, $n \geq 2,$ can be represented in the form

$$F(x_1,\cdots,x_n) = \sum_{j=1}^{2n+1} h_j \left(\sum_{i=1}^{n} \phi_{ij}(x_i) \right) \tag{5}$$

where h_j and ϕ_{ij} are continuous functions of one variable, and ϕ_{ij} are monotonically increasing functions independent of F.

According to Kolmogorov's theorem, a continuous multivariate function on a compact set can be expressed using superpositions and compositions of a finite number of single-variable functions.

Based on Kolmogorov's theorem, Hecht-Nielsen provided a theorem that is directly related to neural networks, and is stated below.

Theorem 3 (Hecht-Nielsen): Any continuous real-valued mapping $F:[0,1]^n \rightarrow R^m$ can be approximated to any degree of accuracy by an FNN with n input nodes, 2n+1 hidden units, and k output units.

In continuation to this, the Weierstrass theorem asserts that any continuous real-valued multivariate function can be approximated to any accuracy using a polynomial. The Stone-Weierstrass theorem is a generalization of the Weierstrass theorem, and is usually used for verifying a model's approximation capability to dynamic systems, which is stated as below.

Theorem 4 (Stone-Weierstrass): Let F be a set of real continuous functions on a compact domain u of n dimensions. Let F satisfy the following criteria

1. **Algebraic closure:** F is closed under addition, multiplication, and scalar multiplication. That is, for any two $f_1, f_2 \in F$, we have $f_1 f_2 \in F$ and $a_1 f_1 + a_2 f_2 \in F$, where a_1 and a_2 are any real numbers.
2. **Separability on u:** for any two different points $x_1, x_2 \in u$, $x_1 \neq x_2$, there exists $f \in F$ such that $f(x_1) \neq f(x_2)$.
3. **Not constantly zero on u:** for each $x \in u$, there exists $f \in F$ such that $f(x) \neq 0$.

Then F is a dense subset of c(u), the set of all continuous real-valued functions on u. In other words, for any $\varepsilon > 0$ any function $g \in C(u)$, there exists $f \in F$ such that $|g(x) - f(x)| < \varepsilon$ for any $x \in u$.

4. Universal approximation

Universal approximation to a given nonlinear functional under certain conditions can be realized by using the classical Volterra series or the Wiener series. The MLP is also a universal approximator and has a strong classification capability. The universal approximation capability of the MLP stems from the nonlinearities used in the nodes. A four-layer MLP can approximate any non-linear relation to any accuracy as long as the number of nodes in the hidden layers are sufficiently large. According to Kolmogorov's theorem, a four-layer MLP with N(2N + 1) nodes using continuously increasing non-linearities can compute any continuous function of N variables. In, a four-layer MLP with $\frac{n}{2}+3$ hidden neurons has been designed and it can represent M distinct examples, with negligibly small error. A four-layer MLP with $2\sqrt{(k+2)M}$ hidden neurons can learn any distinct examples with any arbitrarily small error, where k is the required number of output neurons.

The MLP is very efficient for function approximation in high-dimensional spaces. The error convergence rate of the MLP is independent of the input dimensionality, while conventional linear regression methods suffer from the curse of dimensionality, which results in a decrease of the convergence rate with an increase of the input dimensionality. The necessary number of MLP neurons for approximating a target function depends only upon the basic geometrical shape of the target function, and not on the dimensionality of the input space. Based on geometrical interpretation of the MLP, the minimal number of line segments or hyper planes that can construct the basic geometrical shape of the target function is suggested as the first trial for the number of extrema and the number of hidden nodes should be selected as the number of extrema.

Since both the three-layer and four-layer MLPs can be used as universal approximators, one may wonder as to which one is more effective. Usually, a four-layer network can approximate the target with fewer connection weights, but this may, however, introduce extra local minima. The geometrical interpretation of the MLP on the basis of the special geometrical shape of the activation function can also be found. For the target function with a flat surface located in the domain, a small four-layer MLP can generate better results.

5. Sigma-Pi Networks

The Sigma-Pi network is a generalization of the MLP. Unlike the MLP, the Sigma-Pi network uses product units as well as summation units to build higher-order terms. The BP learning rule can be applied to the learning of the network. The Sigma-Pi network is known to provide inherently more powerful mapping capabilities than first-

order models such as the MLP. It is a universal approximator. However, the Sigma-Pi network has a combinatorial increase in the number of product terms and weights. This limits its applications.

Example 1: Solve the network to approximate the function $f(x) = 1 + \sin \pi x$ for $-1 \le x \le 1$, choosing initial weights and bias as the random numbers, using back propagation algorithm.

Solutions: Choosing for three layer, 1-2-1 network is one neuron in input layer, two neurons in hidden layer and one neuron in output layer, with the weighted structure given as

$$[W]^{\circ} = \begin{bmatrix} -0.25 \\ -0.40 \end{bmatrix}, \text{ bias } \phi^{(1)}(0) = \begin{bmatrix} -0.50 \\ -0.1 \end{bmatrix}$$

and $[V]^{\circ} = [0.1, \quad -0.2]$, bias $\phi^{(2)}(0) = \{0.5\}$

the architecture of this model is given in Fig. 4.

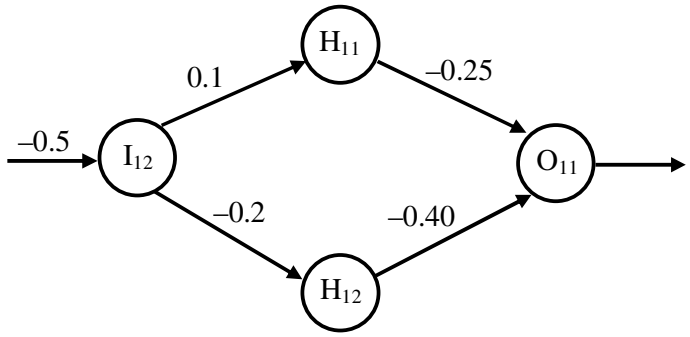


Fig. 4

Let the initial input be 1, then the output of the first hidden layer is given by

$$[O_1]_H = f^{(1)}\left([W]^{\circ} \cdot [O]^{(0)} + \phi^{(1)}\right)$$

$$= f^{(1)}\left(\begin{bmatrix} -0.25 \\ -0.40 \end{bmatrix}[1] + \begin{bmatrix} -0.5 \\ -0.1 \end{bmatrix}\right)$$

$$= f^{(1)}\begin{pmatrix} -0.75 \\ -0.5 \end{pmatrix}$$

$$= \frac{1}{1+e^{0.75}} \\ \frac{1}{1+e^{0.5}}$$

$$[O_2]_H = f^{(2)}\left([V]^{\circ}[O_1]_H + \phi^{(2)}\right)$$

$$= f^{(2)}\left([0.1 \quad -0.2]\begin{bmatrix} 0.321 \\ 0.358 \end{bmatrix} + [0.5]\right)$$

$$= f^{(2)}(0.115) = 0.115 \quad (\text{selecting f as linear function})$$

$$\text{The error} = O_{TO} - [O_2]_H$$

$$= 1 + \sin \pi x - 0.115$$

$$= 0.835$$

Further $[\Delta W]$ and $[\Delta V]'$ can be found and accordingly modified weights can be calculated. The Bias is calculated as given below

$$\phi^{(1)}(1) = \phi^{(1)}(0) - \eta\, s^{(1)}$$

$$\phi^{(2)}(1) = \phi^{(2)}(0) - \eta\, s^{(2)}$$

where $s^{(1)} = F^{(1)} V^t s^{(2)}$ [Here $F'^{(1)}$ is a matrix of $f'^{(1)}$ for two rows of vector]

$s^{(2)} = -f'^{(2)}[O_2]_H$ and η is the learning rate.

The first iteration of backpropagation algorithm is complete here. The another iteration can be perform by choosing another input. The process can be continued to iterate until the difference between the target function and the network response reaches at some acceptable level.

Now try the following exercise.

E1) Solve the network to approximate the function $g(x) = 1 + \sin(\pi x/4)$ for $-2 \le x \le 2$, choosing initial weights and bias as the random numbers.

E2) In a 3-layer backpropagation network, if initial weights and biases are choosen as: $W = -1, \phi^{(1)} = 1, V = -2$, $\phi^{(2)} = 1$ and an input/target pair is given to be $(I = -1, t = 1)$, perform one iteration of backpropagation with $\eta = 1$.

E3) Show that a multi-layer network with linear transfer functions is equivalent to a single-layer linear network.

E4) Show that backpropagation reduces to the LMS algorithm for a single-layer linear network (ADALINE).

E5) Train a BP network to approximate the plot of the function $f(x) = \sin 2\pi x$ over the interval $0 \le x \le 1$.

E6) Train BP network so that it is capable of approximating the plot of the function $f = x^2 + y^2$ over the interval $x \in (0,1)$ and $y \in (0,1)$.

In the following section, we shall discuss generalization.

7.3 GENERALIZATION

According to the approximation of function, learning can be defined as a hyper surface reconstruction based on existing examples, while generalization is estimating the value on the hyper surface where there is no example. In approximation, an example is always required but in generalization examples is not needed. Mathematically, the learning process is a nonlinear curve-fitting process, while generalization is the interpolation and extrapolation of the input data. Neural network have the capabilities of learning and generalization.

If an input always generates a unique output, and the mapping is continuous, the problem of reconstructing the mapping is said to be well-posed. Learning is an ill-posed inverse problem. An approximate solution is required to be found for the mapping in the given example of an input-output mapping. The input data may be noisy or imprecise, and also may be insufficient to uniquely construct the mapping. Here an ill-posed problem is transformed into a well-posed one so as to stabilize the

solution by adding some auxiliary non-negative functional that embeds prior information such as the smoothness constrained for approximation.

When a network is over trained with too many examples, parameters or epochs, it may produce good results for the training data, but has a poor generalization capability. This is the over fitting phenomenon, and is illustrated in Fig. 5. In statistics, over fitting applies to the situation wherein a model possesses too many parameters, and fits the noise in the data rather than the underlying function. A simple network with smooth input-output mapping usually has a better generalization capability. Generally, the generalization capability of a network is jointly determined by the size of the training pattern set, the complexity of the problem, and the architecture of the network.

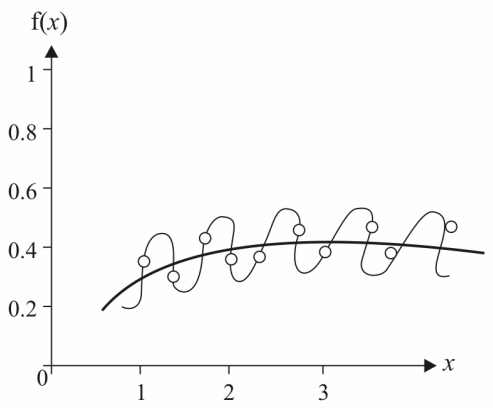


Fig. 5: Proper fitting and over fitting

In Fig. 5, the solid line corresponds to proper fitting, and dashed line corresponds to over fitting.

Size of Training Set

For a given network topology, we can estimate the minimal size of the training set for successfully training the network. For conventional curve-fitting technique, the required number of examples usually grows with the dimensionality of the input space, namely, the curse of dimensionality. Feature extraction can reduce input dimensionality and thus improve the generalization capability of the network.

The training set should be sufficiently large and diverse so that it could represent the problem well. For good generalization, the size of the training set should be at least several times larger than the network's capacity, i.e. $N >> \frac{N_w}{N_y}$, where N is the size of training set, N_w the total number of weights or free parameters, and N_y the number of output components.

Generalization Error

The generalization error of a trained network can be computed by errors, one is approximation error (AE) and the other is estimation error (EE) two. An approximation error that is due to the finite number of parameters of the approximation scheme used, and an estimation error that is due to the finite number of data available.

For an forward neural network with I input nodes and a single output node, a bound for the generalization error is (GE) given by

$$GE = AE + EE$$

$$= O\left(\frac{1}{P}\right) + O\left(\left[\frac{PI\ln(PN) - \ln\delta}{N}\right]^{1/2}\right) \quad (6)$$

With a probability greater than $1-\delta$, where N is the number of examples, $\delta \in (0,1)$ is the confidence parameter, and P is proportional to the number of parameters, such as P sigmoid hidden units or P threshold units in an MLP.

A bound on the total generalization error is a function of the order of the number of hypothesis parameters P and the number of examples N. The approximation error decreases as P, the order of the model, increases. Since we are using a larger model; however, the estimation error increases due to over fitting (or alternatively, more data). Thus, one cannot reduce the upper bounds on both the error components simultaneously. When P is selected as

$$P \alpha N^{\frac{1}{3}} \quad (7)$$

then the tradeoff between the approximation and estimation errors is best maintained, and this is the optimal size of the model for the amount of data available. After suitable selecting P and N, the generalization error for FNNs should be $O\left(\frac{1}{P}\right)$. This result is similar to that for an MLP with sigmoid functions.

The objective of learning is to achieve good generalization to new cases, otherwise just use a look-up table. Generalization can be defined as a mathematical interpolation or regression over a set of training points:

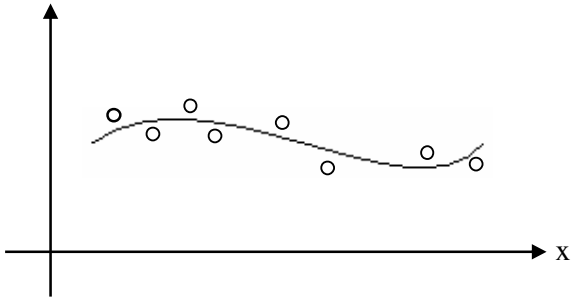


Fig. 6: A set of training points

Example 2 (Computing Parity): The network given in Fig. 7 can learn from m examples to generalize to all 2^n possibilities?

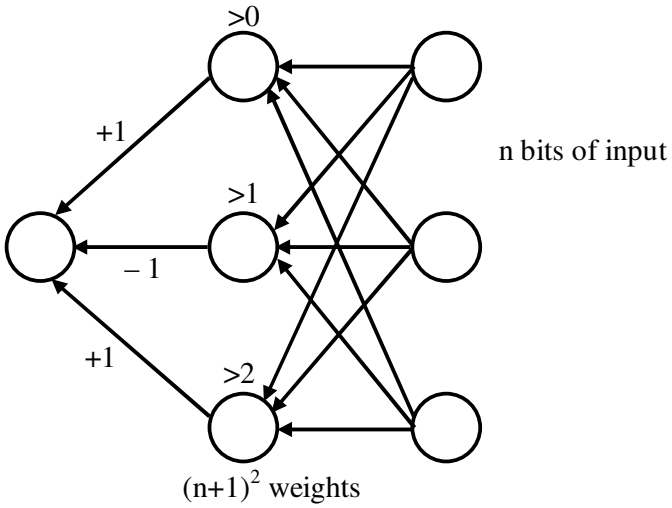


Fig. 7: 2^n possible examples

If we look at many of the problems that have been considered in the field of machine learning, we notice that in all the cases an instance of the problem is given by a set D of input-output pairs, $D \equiv \{(x_i, y_i) \in X \times Y\}_{i=1}^{N}$, belonging to some input and output space.

We assume that there is some relationship between the input and the output, and consider the pairs (x_i, y_i) as examples of it. Therefore we assume that there exists a map

$$f : X \to Y$$

with the property:

$$f(x_i) = y_i \quad i = 1, \cdots, N.$$

Learning, or generalizing, means being able to estimate the function at points of its domain X where no data are available. From a mathematical point of view this means estimating the function f from the knowledge of a subset D of its graph, which is from a set of sparse data. Therefore from this point of view the problem of learning is equivalent to the problem of surface reconstruction. Of course learning and generalization are possible only under the assumption that the world in which we live is at the appropriate level of description redundant. In terms of surface approximation it means that the surface to be reconstructed have to be smooth: small changes in some input determine a correspondingly small change in the output (it may be necessary in some cases to accept piecewise smoothness). Generalization is not possible if the mapping is completely random. For instance, any number of examples for the mapping represented by a telephone directory (people's names into telephone numbers) do not help in estimating the telephone number corresponding to a new name.

Some examples are in order at this point.

Example 3: Learning to pronounce English words from their spelling.

Solution: A well known neural network implementation has been claimed to solve the problem of converting strings of letters in strings of phonemes (Sejnowski and Rosenberg, 1987). The mapping to be learned was therefore $X \equiv \{\text{English words}\} \to Y \equiv \{\text{phonemes}\}$. The input string consisted of 7 letters, and the output was the phoneme corresponding to the letter in the middle of string. Feeding English text in a window of 7 letters, a string of phonemes was produced and then processed by a digital speech synthesizer, therefore enabling the machine to read the text aloud. Because of the particular way of encoding letters and phonemes, the input consisted of a binary vector of 203 elements, and the output of a vector of 26 elements. Clearly in this case the smoothness assumption is often violated, since similar words do not always have similar pronunciations, and this is reflected in the fact that the percentage error on a test set of data was 78%, for continuous informal speech.

Example 4: Learning to recognize a 3D objects from its 2D image.

Solution: The input data set consists of 2D views of different objects in different poses. The output corresponding to a given input is 1 if the input is a 2D view of the object that has to be recognized, and 0 otherwise. A 2D view is a set of features of the 2D images of the object. In the simplest case a feature is the location, on the image plane, of some reference point, but may be any parameter associated to the image. For example, if we are interested in recognizing faces, common features are nose and mouse width, eyebrows thickness, pupil to eyebrows separation, pupil to nose vertical distance, mouth height. In general the mapping to be learned is:

$$X \equiv \{\text{2D view}\} \to Y \equiv \{0, 1\}.$$

Example 5: Learning Navigation tasks.

Solution: An indoor robot is manually driven through a corridor, while its frontal camera records a sequence of frontal images. Can this sequence be used to train the robot to drive by itself without crashing into the walls by using the visual input and to generalize to slightly different corridors and different positions within them? The map to be approximated is

$$X \equiv \{\text{images}\} \rightarrow Y \equiv \{\text{steering} \quad \text{command}\}.$$

Example 6: Learning Motor Control.

Solution: Consider a multiple-joint robot arm that has to be controlled. One needs to solve the inverse dynamics problem: Compute from position, velocities and acceleration of the joint the corresponding torques at the joints. The map to be learned is, therefore, the "inverse dynamics" map

$$X \equiv \{\text{state space}\} \rightarrow Y \equiv \{\text{torques}\}.$$

where the state space is the set of all admissible position, velocities and accelerations of the robot. For a two-joint arm the state space is six dimensional and there are two torques to be learned, one for each joint.

Example 7: Learning to predict time series.

Solution: Suppose we observe and collect data about the temporal evolution of a multi-dimensional dynamical system from some time in the past up to now. Given the state of the system at some time l in the future we would like to be able to predict its state at a successive time l + T. In practice we usually observe the temporal evolution of one of the variables of the system, say $x(l)$, and represent the state of the system as a vector of "delay variables" (Farmer and Sidorowich, 1987):

$$x(l) \equiv \{x(l),\ x(l-\tau), \ldots, x(l-(d+1)\tau)\}$$

where τ is an appropriate delay time, and d is larger than the dimension of the attractor of the dynamical system. The map that has to be learned is therefore

$$f_T : R^d \rightarrow R,$$
$$f_T(x(l)) \rightarrow x(l+T).$$

Now, let us discuss the selection of architecture.

7.4 SELECTION OF ARCHITECTURE

The objective of model selection is to find a model that is as simple as possible that fits a given data set with sufficient accuracy, and has a good generalization capability to unseen data. The generalization performance of a network gives a measure of the quality of the chosen model. Existing model-selection approaches can be generally grouped into four categories: crossvalidation, complexity criteria, regularization, and network pruning/growing. Some existing model-selection methods have been reviewed in.

In crossvalidation methods, many networks of different complexity are trained and then tested on an independent validation set. The procedure is computationally demanding and/or requires additional data withheld from the total pattern set. In complexity criterion-based methods, training of many networks is required and hence,

computationally demanding, though a validation set is not required. Regularization methods are described in the preceding section; they are more efficient than crossvalidation techniques, but the results may be suboptimal since the penalty terms damage the representation capability of the network. Pruning/growing methods can be under the framework or regularization, which often makes restrictive assumptions, resulting in networks that are suboptimal. In addition, the evolutionary approach is also used to select an optimum architecture of neural networks.

Crossvalidation

Crossvalidation, a standard tool in statistics, is a classical model-selection method. The total pattern set is randomly partitioned into a training set and a validation (test) set. The major part of the total pattern set is included in the training set, which is used to train the network. The remaining, typically, 10 to 20 percent, is included in the validation set and is used for validation. When only one sample is used for validation, the method is called leave-one-out crossvalidation.

Complexity Criteria

An efficient approach for improving the generalization performance is to construct a small network using a parsimonious principle. Statistical model selection with information criteria such as Akaike's final prediction error (FPE) criterion, Akaike information criterion(AIC), Schwartz's Bayesian information criterion (BIC), and Rissanen's minimum description length (MDL) principle are popular and have been widely used for model selection of neural networks.

The selection of architecture is done with the help of following aspects.

Design:

- Architecture of network
- Structure of artificial neurons
- Learning rules

Training:

- Ensuring optimum training
- Learning parameters
- Data preparation
- and more…

Now let us discuss every aspect in detail.

Architecture of the network: First of all we determine the number of network weights, the number of layers, and the number of nodes per layer.

The automated methods like augmentation (cascade correlation) and weight pruning and elimination are applied to find the architecture of the network.

Now for the connectivity of the Architecture of the network the model or hypothesis space are conceptualized. The number of hypotheses are constrained by selective connectivity, shared weights and recursive connections.

Once architecture is finalized, then the structure of artificial neuron nodes is processed by the choice of input integration and choice of activation (transfer) function.

Then a learning rule is selected from the following list:

1. Generalized delta rule (steepest descent)
2. Momentum descent
3. Advanced weight space search techniques
4. Global Error function can also vary

   - normal -quadratic - cubic

With this the designing part of the network is done. Now the training aspect of a neural network for training the following steps are followed:

1. Establish a maximum acceptable error rate.
2. Train the network using a validation test set to tune it.
3. Validate the trained network against a separate test set which is usually referred to as a production test set.

Now try the following exercise.

E7) Discuss the architecture of a neural network.

Now, let us summarize the unit.

7.5 SUMMARY

In this unit, we have covered the following points.

1. The function approximation lays the theoretical foundation for the wide application of forward neural networks.
2. A trained network represents the learned non-linear relation between the input-output pairs, and can generalize where an unknown input pattern is presented to the network.
3. The selection of architecture is done to find a model that is as simple as possible that fits a given data set with sufficient accuracy, and has a good generalization capability to unseen data.

7.6 SOLUTIONS/ANSWERS

E1) Choosing for a 1-2-1 network, a weight structure of

$$W(0)=\begin{bmatrix}-0.27\\-0.41\end{bmatrix},\ \phi^{(1)}(0)=\begin{bmatrix}-0.48\\-0.13\end{bmatrix}$$

and $V(0)=[0.09-0.17],\ \phi^{(2)}(0)=\{0.48\}$

Let initial input be $a^{(0)}=1$, the output of the first-layer (hidden) is then:

$$a^{(1)}=f^{(1)}\left(W\cdot a^{(0)}+\phi^{(1)}\right)$$

$$=f^{(1)}\left(\begin{bmatrix}-0.27\\-0.41\end{bmatrix}[1]+\begin{bmatrix}-0.48\\-0.13\end{bmatrix}\right)$$

$$=f^{(1)}\begin{pmatrix}-0.75\\-0.54\end{pmatrix}=\begin{bmatrix}1/1+e^{0.75}\\1/1+e^{0.54}\end{bmatrix}=\begin{bmatrix}0.321\\0.368\end{bmatrix}$$

The second-layer output is:

$$a^{(2)} = f^{(2)}\left(V \cdot a^{(1)} + \phi^{(2)}\right)$$

$$= f^{(2)}\left(\left[0.09 - 0.17\begin{bmatrix}0.321\\0.368\end{bmatrix} + 0.48\right]\right)$$

$$= f^{(2)}(0.446) = 0.446 \quad \left(\text{by choosing } f^{(2)} \text{ as linear function}\right)$$

The error would then be

$$e = t - a^{(2)} = \left[1 + \sin\frac{\pi}{4}x\right] - a^{(2)} = \left[1 + \sin\frac{\pi}{4}\right] - 0.446 = 1.261$$

Next step of algorithm is to backpropagate the sensitivities. Before, beginning backpropagation: recalling that the derivatives of transfer function are needed.

We have for the first-layer

$$f'^{(1)}(x) = \frac{d}{dx}\left[\frac{1}{1+e^{-x}}\right] = \left(1 - a^{(1)}\right) \cdot a^{(1)}$$

For the second layer: $f'^{(2)}(x) = \frac{d}{dx}x = 1.$

Starting point is found at the second layer:

$$V(1) = V(0) - \eta \cdot s^{(2)} \cdot \left(a^{(1)}\right)^T$$

$$= V(0) - 0.2 \cdot \left[-f'^{(2)}\left(a^{(2)}\right)\right] \cdot (t - a) \cdot \left(a^{(1)}\right)^T$$

$$= [0.09 \quad -0.17] + 0.2 \cdot [1] \cdot (1.261) \cdot [0.321 \quad 0.368]$$

$$= [0.171 \quad -0.07721]$$

and $W(1) = W(0) - \eta \cdot s^{(1)} \cdot \left(a^{(0)}\right)^T$

where $s^{(1)} = F'^{(1)} \cdot V^T \cdot s^{(2)}$, where $F'^{(1)}$ is a matrix of $f'^{(1)}$ for two rows of vector.

$$= \begin{bmatrix}\left(1 - a_1^{(1)}\right) \cdot a_1^{(1)} & 0\\ 0 & \left(1 - a_2^{(1)}\right) \cdot a_2^{(1)}\end{bmatrix} \cdot \begin{bmatrix}0.09\\-0.17\end{bmatrix} \cdot [-0.2 \, x \, 1.261]$$

$$= \begin{bmatrix}-0.0495\\0.0997\end{bmatrix}$$

$$\therefore \; W(1) = W(0) - \eta \cdot s^{(1)} \cdot \left(a^{(0)}\right)^T = \begin{bmatrix}-0.27\\-0.41\end{bmatrix} - 0.2\begin{bmatrix}-0.0495\\0.0997\end{bmatrix} = \begin{bmatrix}-0.260\\-0.429\end{bmatrix}$$

Bias is likewise calculated:

$$\phi^{(2)}(1) = \phi^{(2)}(0) - \eta s^{(2)} = 0.48 - 0.2 \times [-2.522] = 0.9844$$

$$\phi^{(1)}(1) = \phi^{(1)}(0) - \eta s^{(1)} = \begin{bmatrix}-0.48\\-0.13\end{bmatrix} - 0.2\begin{bmatrix}-0.0495\\0.0997\end{bmatrix} = \begin{bmatrix}-0.4701\\-0.1499\end{bmatrix}$$

This completes one iteration of backpropagation algorithm. Next proceed to

choose another input and perform another iteration. Continue to iterate until the difference between network response and the target function reaches some acceptable level.

E2) First step is to propagate the input through network:

$$n^{(1)} = W \cdot I - + \phi^{(1)} = (-1)\cdot(-1) + 1 = 2$$

Hence the activation $a^{(1)} = \tanh(2) = 0.964$

Now $n^{(2)} = V \cdot a(1) + \phi^{(2)} = -2(0.964) + 1 = -0.928$

Hence its activation $a^{(2)} = \tanh\left(n^{(2)}\right) = \tanh(-0.928) = -0.7297$

Therefore error

Now, backpropagation sensitivities:

$$s^{(2)} = -1 \cdot f'^{(2)}\left(n^{(2)}\right)\cdot\left(t - a^{(2)}\right)$$

$$= -1 \cdot \left[1 - \left(a^{(2)}\right)^2\right] \times 1.7297$$

$$= -\left[1 - 0.7297^2\right] \times 1.7297 = -0.8087$$

and $s(1) = f'^{(1)}\left(n^{(1)}\right) V^T \cdot s^{(2)}$,

$$= \left[1 - \left(a^{(1)}\right)^2\right] \times V^T \cdot s^{(2)} = \left[1 - 0.964^2\right]\cdot(-2)\cdot(-0.8087)$$

$$= 0.1142$$

Finally, the weights and biases are updated using the equations:

$$V(1) = V(0) - \eta \cdot s^{(2)} \cdot \left(a^{(1)}\right)^T = -2 - 1 \times (-0.8087)\cdot(0.964) = -1.2204$$

$$W(1) = W(0) - \eta \cdot s^{(1)} \cdot \left(a^{(0)}\right)^T = -1 - 1 \times (0.1142)(-1) = -0.8858.$$

$$\phi^{(1)}(1) = \phi^{(1)}(0) - \eta s^{(1)} = 1 - 1 \times 0.1142 = 0.8858$$

$$\phi^{(2)}(1) = \phi^{(2)}(0) - \eta s^{(2)} - 1 - 1 \times (-0.8087) = 1.8087.$$

E3) For a multi-layer network, the forward equations are:

$$a^{(1)} = W^{(1)} \cdot I + \phi^{(1)}$$

$$a^{(2)} = W^{(2)} \cdot a^{(1)} + \phi^{(2)} = W^{(2)} \cdot W^{(1)} \cdot I + W^{(2)} \cdot \phi^{(1)} + \phi^{(2)}$$

$$a^{(3)} = W^{(3)} \cdot a^{(2)} + \phi^{(3)}$$

$$= \left[W^{(3)} \cdot W^{(2)} \cdot W^{(1)} \cdot I + W^{(3)} \cdot W^{(2)} \cdot \phi^{(1)} + W^{(3)} \cdot \phi^{(2)} + \phi^{(3)}\right]$$

For an M-layer linear network, the equivalent single-layer linear network has weight matrix and bias vector as:

$$W = W^{(M)} \cdot W^{(M-1)} \ldots W^{(1)},$$

and $\phi = \left[W^{(M)} \cdot W^{(M-1)} \cdots W^{(2)}\right]\cdot\phi^{(1)}\Big] + W^{M)} \cdot W^{M-1)} \cdots W^{(3)}\Big]\cdot\phi^{(2)} + \ldots + \phi^{(M)}$.

E4) Sensitivity to be backpropagated for single-layer linear network is:

$$s^{(1)} = -f'^{(1)}\left(n^{(1)}\right)\cdot(t - a) = -e(\text{since } f'(x) = 1)$$

The weight update is given by

$$W(k+1) = W(k) - \eta \cdot s^{(1)} \cdot a\left(a^{(0)}\right)^T$$

$$= W(k) + \eta.e.(I)^T$$

and bias is $\phi(k+1) = \phi(k) - \eta s^{(1)} = \phi(k) + \eta e$

This is identical to LMS algorithm (ADALINE).

UNIT 8 RADIAL BASIS FUNCTION NETWORKS

Structure **Page No.**

8.1 INTRODUCTION

We now move on to discuss a class of feed forward neural networks called radial basis function networks (RBFN) that compute activations at the hidden neurons in a way that is different from what we have seen in the case of feed forward neural networks. RBFNs are computed using an exponential of a distance measure between the input vector and a prototype vector that characterizes the signal function at a hidden neuron. We shall discuss RBFN in Sec. 8.2. We shall continue calculations of these networks for Backward and forward propagation in Sec. 8.3. At last in Sec. 8.4 we shall discuss some application of there networks.

Objectives

After studying this unit you should be able to:

- define the radial basis function networks;
- calculate the modified weight using backward and forward propagation;
- define the applications.

8.2 RADIAL BASIS FUNCTION NETWORK (RBFN)

A radial basis function network is a neural network approached by viewing the design as a curve-fitting (approximation) problem in a high dimensional space. In a neural network, the hidden units form a set of "functions" that compose a random "basis" for the input patterns (vectors). These functions are called radial basis functions.

A radial basis function (RBF) is a special type of function, whose response decreases or increases monotonically with its distance from a central point. Various types of RBFs are given below:

1. **Thin plate spline function:** $f(x) = x^2 \log(x)$,
2. **Gaussian function:** $f(x) = \exp\left(-\frac{\|x-\mu\|^2}{2\sigma^2}\right)$, where μ and σ indicate the mean and standard deviation of the distribution, respectively,
3. **Multi-quadratic function:** $f(x) = \sqrt{x^2 + \sigma^2}$,
4. **Inverse multi-quadratic function:** $f(x) = \frac{1}{\sqrt{x^2 + \sigma^2}}$.

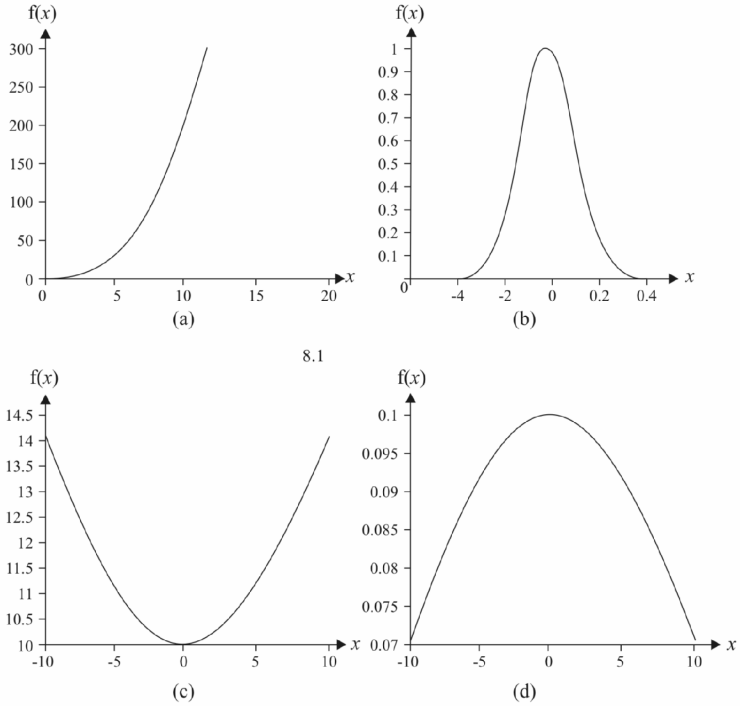


Fig. 1: Graphs various radial basis function

Fig. 1 shows the plots of the above RBFs, (a): Plate spline function, (b): Gaussian function, (c): Multi-quadratic function and (d): Inverse multi-quadratic function.

In 1988, Broomhead and Lowe proposed Radial Basis Function Network (RBFN). It is a special type of Artificial Neural Network (ANN) that can fit a surface in multi-dimensional space after ensuring the best match to the training data. These networks can tackle function approximation, the problems related to time-series forecasting, classification, clustering, recognition of handwritten characters, face recognition, voice identification and others. This network consists of an input layer, hidden layer and output layer. The input layer consists of some input data nodes. The number of these nodes is equal to the number of independent input variables of the system to be modeled. The hidden layer of a few neurons is with radial basis transfer functions. The output layer composed of some neurons (number is made equal to the number with linear transfer function usually, and the number of outputs}. The performance of the RBFN is dependent on the number of hidden neurons, their centers and radii. For example, the performance of a network with Gaussian transfer functions depends on their centers and radii, which are nothing but their means and standard deviations, respectively. The network with an insufficient number of hidden neurons may not be able to offer good approximation. On the other hand, a large number of hidden neurons may yield good approximation but at the cost of predictive power. It is known as an **over-fitting problem** of the network. The RBFN with either narrow or wide radii may produce some bad results. Thus, a proper tuning of the network is necessary to carry out.

RBFN are usually trained by the following methods:

i) The number of hidden neurons is pre-defined and suitable values of their centers and radii are determined through a proper training.

ii) The algorithm starts with an empty set of hidden neurons, which grow in number until a given condition is reached. Thus, the number of hidden neurons and their centers and radii are searched iteratively.

iii) The algorithm begins with a large number of hidden neurons. Some of the hidden neurons and their connecting weights are removed iteratively and

ultimately, the optimal network is obtained.

The input-output relationship of a RBFN with N independent input variables and M output is shown in Fig. 2.

Let us suppose that the network consists of an input layer having N input nodes, one hidden layer containing P neurons and an output layer of M neurons. Let us also assume that a set of L training scenarios will be used to train the network. For a particular training scenario (say-l-th), the forward and backward propagation are discussed in the following section.

8.3 FORWARD AND BACKWARD PROPAGATION

The following steps are considered in the forward calculations:

i) The outputs of input layer

Let us represent L training scenarios, which constitute the input vector, as given

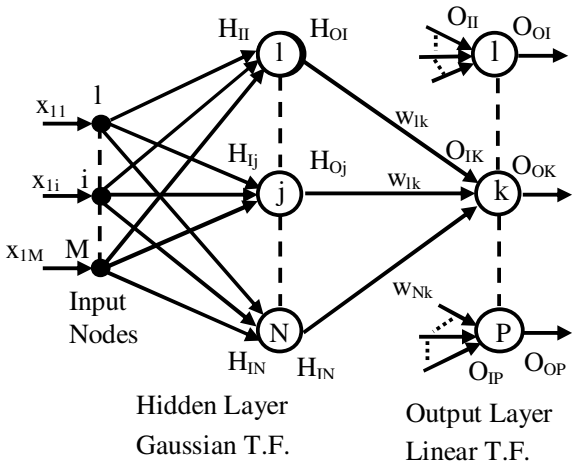


Fig. 2: Radial Basis Function Network.

below:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_l \\ \vdots \\ x_L \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1i} & \cdots & x_{1N} \\ x_{21} & x_{22} & \cdots & x_{2i} & \cdots & x_{2N} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{l1} & x_{12} & \cdots & x_{li} & \cdots & x_{lN} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{L1} & x_{L2} & \cdots & x_{Li} & \cdots & x_{LN} \end{bmatrix}$$

Let us assume that 1-th training scenario (that is, $x_{l1}, x_{l2,} \ldots, x_{li,} \ldots, x_{lM}$ expressed in the normalized form) is passed through the network. The outputs of different nodes of the input layer are nothing but the inputs of corresponding nodes.

ii) The outputs of hidden layer

The hidden layer consists of p neurons. Each of them has a Gaussian transfer function represented by a separate set of mean μ and standard deviation σ values. Thus, the values of μ and σ associated with the transfer functions of 1-st, 2-nd, …, p-th neurons are represented by $(\mu_1, \sigma_1), (\mu_1, \sigma_2), \cdots, (\mu_p, \sigma_p)$,

respectively. The output of j-th hidden neuron can be expressed like the following:

$$[O_H]_j = \exp\left[-\frac{\|x_1 - \mu_j\|^2}{2\sigma_j^2}\right] \quad (1)$$

iii) The inputs of output layer

The input of k-th neuron lying on output layer can be represented as follows:

$$[I_O]_k = \sum_{j=1}^{N} w_{Jk}\, O_{Hj} \quad (2)$$

where $w_{jk`}$ indicates the connecting weight between j-th hidden neuron and k-th output neuron.

iv) The output of output layer

The output of k-th neuron lying on output layer is determined as follows:

$$[O_O]_k = [I_O]_k \quad (3)$$

here the output neurons are assumed to have linear transfer function.

Let us suppose that the target output of k-th output neuron is denoted by T_{Ok}. Therefore, the error in prediction can be calculated like the following:

$$E_k = \frac{1}{2}(T_{Ok} - O_{Ok})^2. \quad (4)$$

Back-propagation algorithm can be implemented through either an incremental or a batch mode of training as discussed below.

Let us consider an incremental mode of training, that is, the connecting weight, mean and standard deviation values of the Gaussian distribution will be updated to minimize the error in prediction, corresponding to l-th training scenario.

i) Weight updating

The weights are updated following the rule given below.

$$w_{t+1} = w_t + \Delta w, \quad (5)$$

where Δw indicates the change in w value. The change in w value at t-th iteration is given by the following expression:

$$\Delta w_{jk}(t) = -\eta \frac{\partial E_k}{\partial w_{jk}}(t) + \alpha' \Delta w_{jk}(t-1), \quad (6)$$

where η and α' represent the learning rate and momentum constant, respectively. Now, $\frac{\partial E_k}{\partial w_{jk}}$ can be determined corresponding to k-th output neuron, using the chain rule of differentiation as given below.

$$\frac{\partial E_k}{\partial w_{jk}} = \frac{\partial E_k}{\partial O_{Ok}} \frac{\partial O_{Ok}}{I_{Ok}} \frac{\partial I_{Ok}}{\partial w_{jk}}, \tag{7}$$

where $\frac{\partial E_k}{\partial O_{Ok}} = -(T_{Ok} - O_{Ok})$,

$\frac{\partial O_{Ok}}{\partial O_{Ik}} = 1$,

$\frac{\partial O_{Ik}}{\partial w_{jk}} = O_{Hj}$

Therefore, Eq. (7) can be re-written as

$$\frac{\partial E_k}{\partial w_{jk}} = -(T_{Ok} - O_{Ok}) O_{Hj} \tag{8}$$

Now, Eq. (5) can be expressed like the following:

$$w_{jk,\,updated} = w_{jk,\,previous} + \eta(T_{Ok} - O_{Ok}) H_{Oj} + \alpha' \Delta w_{jk,\,previous.} \tag{9}$$

The similar procedure can be followed to determine updated values of other weights.

ii) Mean updating

The mean of Gaussian distribution corresponding to j-th hidden neuron can be updated as follows:

$$\mu_{j,\,updated} = \mu_{j,\,previous} + \Delta\mu_j, \tag{10}$$

where the change in μ at t-th iteration can be determined like the following:

$$\Delta\mu_j(t) = -\eta \left\{ \frac{\partial E}{\partial \mu_j}(t) \right\} av + \alpha' \Delta\mu_j(t-1), \tag{11}$$

where $\left\{ \frac{\partial E}{\partial \mu_j} \right\} av = \frac{1}{p} \sum_{k=1}^{P} \frac{\partial E_k}{\partial \mu_j}$

$$\frac{\partial E_k}{\partial \mu_j} = \frac{\partial E_k}{\partial O_{Ok}} \frac{\partial O_{Ok}}{\partial I_{Ok}} \frac{\partial I_{Ok}}{\partial O_{Hj}} \frac{\partial O_{Hj}}{\partial \mu_j} \tag{12}$$

We have already seen that $\frac{\partial E_k}{\partial O_{Ok}} \frac{\partial O_{Ok}}{\partial I_{Ok}} = -(T_{Ok} - O_{Ok})$.

Now, $\frac{\partial I_{Ok}}{\partial O_{Hj}} = w_{jk,}$

$$\frac{\partial O_{Hj}}{\partial \mu_j} = O_{Hj} \left\{ \frac{(x_{l1} + x_{12} + \cdots + x_{1i} + \cdots + x_{lM}) - M\mu_j}{\sigma_j^2} \right\}.$$

Therefore, we get

$$\frac{\partial E_k}{\partial \mu_j} = -(T_{Ok} - O_{Ok}) w_{jk} O_{Hj} \left\{ \frac{(x_{l1} + x_{l2} + \cdots + x_{li} + \cdots + x_{lM}) - M\mu_j}{\sigma_j^2} \right\}. \quad (13)$$

iii) Standard Deviation Updating

The standard deviation corresponding to j-th hidden neuron will be updated as follows:

$$\sigma_{j,\text{updated}} = \sigma_{j,\text{previous}} + \Delta\sigma_j, \quad (14)$$

where the change in σ at i-th iteration is written like the following:

$$\Delta\sigma_j(t) = -\eta \left\{ \frac{\partial E}{\partial \sigma_j}(t) \right\} av + \alpha' \Delta\sigma_j(t-1), \quad (15)$$

where $\left\{ \frac{\partial E}{\partial \sigma_j} \right\} av = \frac{1}{P} \sum_{k=1}^{P} \frac{\partial E_k}{\partial \sigma_j}$,

$$\frac{\partial E_k}{\partial \sigma_j} = \frac{\partial E_k}{\partial O_{Ok}} \frac{\partial O_{Ok}}{\partial I_{Ok}} \frac{\partial I_{Ok}}{\partial O_{Hj}} \frac{\partial O_{Hj}}{\partial \sigma_j}. \quad (16)$$

We have already seen that $\frac{\partial E_k}{\partial O_{Ok}} \frac{\partial O_{Ok}}{\partial I_{Ok}} = -(t_{Ok} - O_{Ok})$.

Now, $\frac{\partial I_{Ok}}{\partial O_{Hj}} = w_{jk}$,

$$\frac{\partial O_{Hj}}{\partial \sigma_j} = O_{Hj} \left\{ \frac{(x_{l1} - \mu_j)^2 + (x_{l2} - \mu_j)^2 + \cdots + (x_{li} - \mu_j)^2 + \cdots + (x_{lM} - \mu_j)^2}{\sigma_j^3} \right\}.$$

Therefore, we get

$$\frac{\partial E_k}{\partial \sigma_j} = -(T_{Ok} - O_{Ok}) w_{jk} O_{Hj} \left\{ \frac{\sum_{i=1}^{M} (x_{li} - \mu_j)^2}{\sigma_j^3} \right\}. \quad (17)$$

8.4 APPLICATIONS

RBNF can be applied for the following

i) Polynomial Fitting

For comparison purpose, Fig. 3 shows the results obtained by the following four methods; linear interpolation, cubic spline interpolation, fourth-order polynomial interpolation, and third-order polynomial approximation. All the methods except for the first one can generate smooth curves. However, it is cumbersome to extend spline and polynomial interpolation to high-dimensional data sets that can be handled by the interpolation RBFN easily.

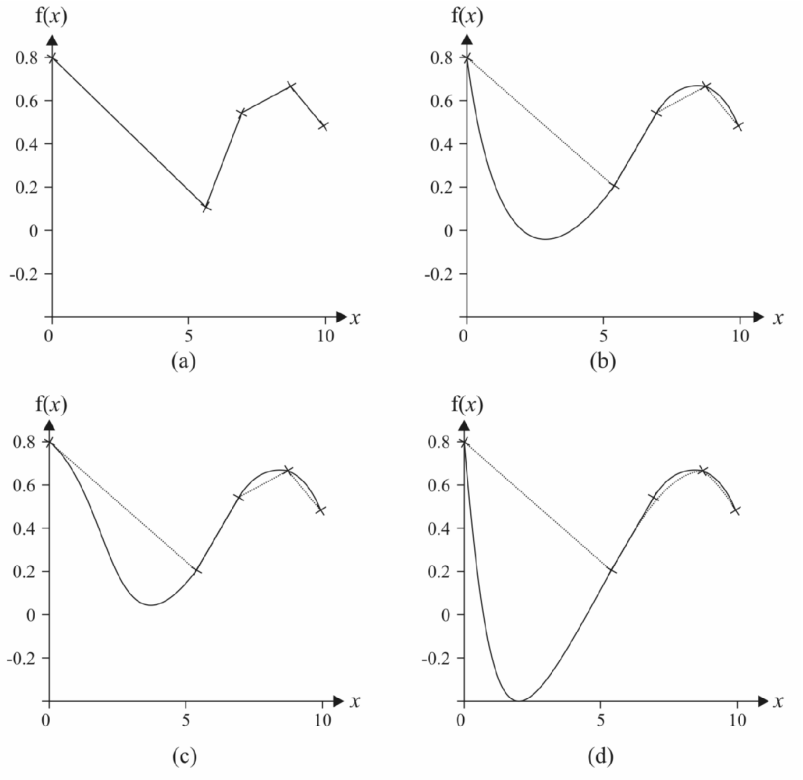


Fig. 3: Interpolation and approximation results obtained by four polynomial interpolation/approximation methods, (a): Linear interpolation, (b): Cubic spine interpolation, (c): Fourth-order polynomial interpolation and (d): Third-order polynomial approximation

ii) Backpropagation MLP Fitting

Fig. 4 presents the results obtained by a backpropagation MLP. In the MLP results, six choices for the number of hidden units are considered to see how the overall responses are affected. The curves generated by MLP are quite smooth and they all pass the five given data points. However, the positions of these curves between data points are strongly influenced by the initial weights of the MLP. Moreover, as more hidden nodes are used, the variations of curves between data points are more pronounced.

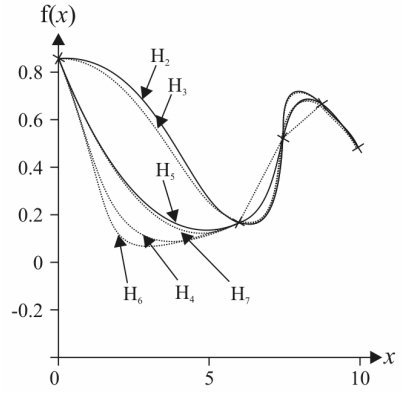


Fig. 4: Interpolation results obtained by simple backpropagation MLPs. "#HU" denoted "number of hidden units"

Now, let us summarize the unit.

8.5 SUMMARY

In this unit we have covered the following points.

1. A radial basis function network is a neural network approached by viewing the design as a curve-fitting (approximation) problem in a high dimensional space.
2. In forward propagation, the outputs of input layer, the output of the hidden layer, the inputs of output layer and the outputs of output layer were determined.
3. In backward propagation, the weights, mean, standard deviations were updated.
4. The application of RBFN.

UNIT 9 HOPFIELD NETWORKS

Structure **Page No.**

9.1 INTRODUCTION

One of the most important functions of human brain is the laying down and recall of memories. More over the studies in neurodynamics indicate the presence of short-term and long-term memory. The short-term memory contributes in processing routine tasks while long-term memory helps in storing the lesson from the past experience. Our memories mostly function in a manner better termed as **associative** or **content-addressable**. That is, a memory does not exist in some isolated fashion, located in a particular set of neurons. All memories are in some sense strings of memories. For example, we remember a place or a person by some event. Sometimes more than the facial features of a person it is the voice or the peculiarity of the name that impresses our memory. Thus, memories are stored in *association* with one another.

The neural network researchers contend that there is a basic mismatch between the standard technology used by computers and the technology of the human brain for processing information. Feed-forward neural networks are well studied for obvious reasons of simplicity in understanding and modelling the computation. Feed forward network is acyclic. The artificial neurons or the perceptrons in a feed-forward neural network model have no cyclic connections. The input data is processed and passed to the outputs but not vive-versa.

Recurrent networks can have connections that go back from the output nodes and can also have arbitrary connections between any nodes. That is a recurrent neural network has at least one feedback loop. For example, in a single layer recurrent network each neuron may have a feedback connection to each of the other neuron or it may even have self-feedback loops.

As learning using perceptrons is a process of modifying the values of the synaptic weights and the threshold or the activation function using samples of training data. A group of (connected) perceptrons are trained on sample input-output pairs until it learns to compute the correct function. Once a feed-forward network is trained, its state is fixed which is not altered by any subsequent input data until the network is retrained. This indicates that a feed-forward neural network does not have memory. In a neurobiological context memory refers to “the relatively enduring neural alterations induced by the interaction of an organism with its environment”. Learning tasks lead us to assume existence of memory. Therefore, the alterations indicate presence of memory and for its usefulness memory needs to be accessible.

Associative memories have a very faint similarity to that of the human brain’s ability to associate patterns. The architecture of a neural network with associative memory

depends on its capability of association. An associate memory is a storehouse of associated patterns encoded in some form. When this storehouse is triggered or incited with a pattern the associated pattern pair is recalled or output. An associative memory network is designed by storing/recording several ideal patterns into the network's stable state. When a pattern (even with noise or distorted representation of stored patterns) is input to the storehouse, the network is expected to reach one of the stored patterns and retrieve it as an output. These neural network models are categorized as association models and are also known as content addressable or auto-associative neural networks. All the definitions will be discussed in Sec. 9.2. In Sec. 9.3 we shall discuss Hopfield networks. In Sec. 9.4, we shall discuss the structure of the Hopfield networks, in detail and we shall extend our discussion on the functionality of the Hopfield networks in Sec. 9.5. The storage capacity of this network is discussed in Sec. 9.6.

Objectives

After reading this unit you should be able to:

- identify and design general association and recurrent models of neural network;
- model Hopfield networks;
- train Hopfield networks for a given set of patterns;
- give a trained network to run the Hopfield for a test pattern with noise may require rigorous computation specially if dimension of the vectors is high.

9.2 RELATED DEFINITIONS

Let us recall various definitions as given below:

Recurrent Networks: The interconnection scheme of the recurrent networks is feedback connection or loops as shown below in Fig. 1.

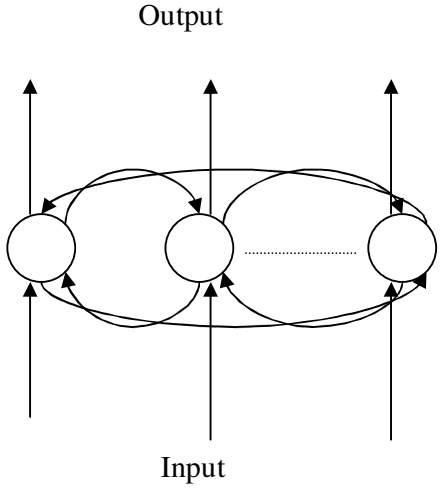


Fig. 1: Recurrent network

Associative Memory: An associative memory is a brain-like distributed memory that learns by association. Association is one of the basic features of the human memory and is prevalent in most models of cognition. Associative memory can be categorized as auto associative memory and Hetero associative memory. These are defined in the following:

i) **Autoassociation**: In autoassociation a neural network is required to store a set of patterns (vectors) by repeatedly presenting them to the network. The network is subsequently presented a partial description or noisy version of an original pattern stored in it, and the task is to recall or more specifically ***retrieve*** that particular pattern.

ii) **Heteroassociation**: In heteroassociation an arbitrary set of input patterns is associated (paired) with another set of arbitrary set of output patterns. The task of retrieval of patterns however is similar i.e. on input of a stored pattern or a distorted version of the already stored pattern the original pattern coupled with the given input is recalled.

Let x_k be the key pattern (vector) applied to an associative memory and y_k be the memorized pattern (vector). The pattern association performed by the network is described by,

$$x_k \rightarrow y_k, k = 1, 2, \ldots, q$$

where q is the number of patterns stored in the network. The key pattern x_k acts as the stimulus that not only determines the storage location of the memorized pattern y_k, but also holds the key for its retrieval.

Fig. 2: Input-output relation of the pattern associator

Auto-associative networks consist of one layer of neural elements (units) that are all interconnected, although self-connections are usually disallowed. The neural elements are nonlinear, with states bounded from zero to one – so called two-state-neurons. Any state, whether initial or desired, will be represented by some pattern of zeros and ones over the units. Recurrence and nonlinearity lie at the heart of auto-associative network behaviour. Recurrence allows desired states to emerge via interactions among the units, and the nonlinearity prevents the whole network from running away, and instead encourages the network to settle into a stable state. The auto-associative network will, in most cases, relax from any initial state into the desired state closest to it.

The interconnections between units can be trained according to Hebbian rules (proposed in 1949) - "*when one cell repeatedly assists in firing another, the axon of the first cell develops synaptic knobs (or enlarges them if they already exist) in contact with the soma of the second cell*." Hebbian rules are *local*, in the sense that they depend only upon the activity of neurons pre- and post-synaptic to any particular connection. The original rule proposed by Hebb specified an increase in synaptic strength whenever pre- and post-synaptic neurons were active together.

Therefore in an association net, if we compare two patterns components (vectors or pixels) within many patterns and find that they are frequently in the same state then the arc weight between the two perceptrons must be increased (should be positive) and if they are frequently in different states, then the arc weight between the two perceptrons must be decreased (should be negative).

There are two phases involved in the operation of an associative memory:

- **Storage phase:** This refers to the training of the network.

- **Recall/Retrieval phase:** This involves the retrieval of a memorized pattern in response to the presentation of a noisy or distorted version of a key pattern to the network.

Matrix Representation: For the computation a matrix representation is a practical tool.

Let X = matrix of input patterns, where each ROW is a pattern.

So $x_{k,i}$ = the i-th bit of the k-th pattern.

Let Y = matrix of output patterns, where each ROW is a pattern. So $y_{k,j}$ = the j-th bit of the k-th pattern. Then, average correlation between two input patterns i and j across all patterns is:

$$w_{i,j} = 1/P(x_{1,i}y_{1,j} + x_{2,i}y_{2,j} + \cdots + x_{p,i}\,y_{p,j}) \tag{1}$$

All weights for an associative memory network model therefore can be calculated by the following:

$$W = X^T Y \tag{2}$$

Therefore, for a set of input patterns: $IP_1, IP_2, \ldots, IP_p$, and output patters $OP_1, OP_2, \ldots, OP_p$,

$$X = \begin{matrix} IP_1 \\ IP_2 \\ \vdots \\ IP_p \end{matrix}\begin{bmatrix} x_{1,1} & \ldots & x_{1,n} \\ x_{2,1} & \ldots & x_{2,n} \\ \vdots & \vdots & \vdots \\ x_{p,1} & \ldots & x_{p,n} \end{bmatrix}$$

$$X^T = \begin{matrix} IP_1 & IP_2 & \ldots & IP_p \end{matrix}\begin{bmatrix} x_{1,1} & x_{2,1} & \ldots & x_{p,1} \\ \vdots & \vdots & \ldots & \vdots \\ x_{1,i} & x_{2,i} & \ldots & x_{p,i} \\ \vdots & \vdots & \ldots & \vdots \\ x_{1,n} & x_{2,n} & \ldots & x_{p,n} \end{bmatrix}$$

$$Y = \begin{matrix} OP_1 \\ OP_2 \\ \vdots \\ OP_p \end{matrix}\begin{bmatrix} y_{1,1} & \ldots\, y_{1,j}\, \ldots & y_{1,n} \\ y_{2,1} & \ldots\, y_{2,j\ldots} & y_{2,n} \\ \vdots & \vdots & \vdots \\ y_{p,1} & \ldots\, y_{p,j\ldots} & y_{p,n} \end{bmatrix}$$

From the above sets of vectors (matrices) X^T and Y, it is obvious that Eqn. (1) is the dot product of i-th row of X^T and the j-th column of Y. But since the output vector in case of autoassociation is one of the earlier memorized vectors it is one of vectors in X itself and therefore the weight calculation autoassociation networks is,

$$W = X^T X \tag{3}$$

Consider the following procedure to explain the associative memory network model.

Input: Pattern (often noisy/corrupt)

Output: Corresponding pattern (complete/relatively noise-free)

Process:

1. Load input pattern onto highly-interconnected neurons (a trained network on given set of library patterns).
2. Run the neurons until they reach a steady state.
3. Read output off the states of the neurons.

Subsequently Fig. 3 and fig. 4 give a graphical illustration of the procedure.

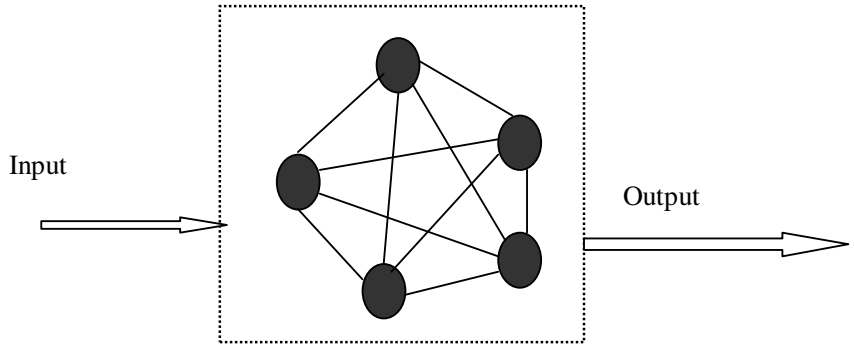


Fig. 3: Trained associative network

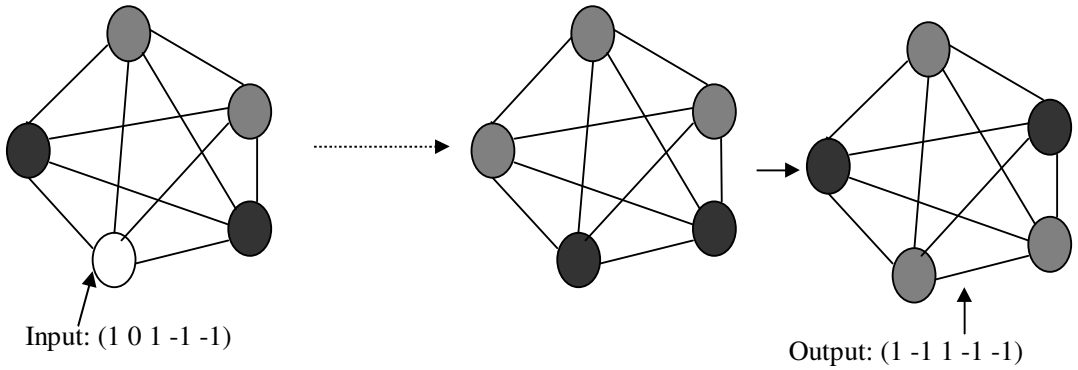


Fig. 4: Running an input pattern

Now, let us discuss Hopfield networks in the following section.

9.3 HOPFIELD NETWORKS

The Hopfield Net proposed by J.J. Hopfield in 1982 is a neural network that is a lot simpler to understand than the Multi Layer Perceptron. For a start, it only has one layer of nodes. Unlike the MLP, it doesn't make one of its outputs go high to show which of the patterns most closely matches the input. Instead it takes the input pattern, which is assumed to be one of the library patterns which has been changed or corrupted somehow, and tries to reconstruct the correct pattern from the input that you give it.

Hopfield networks are Recurrent neural network model that possesses auto-associative property. Therefore the network is fully interconnected with the exception that no neuron has any connection to itself. Thus, the Hopfield network is a form of artificial neural network that serve as content-addressable memory systems with binary threshold units. They are guaranteed to converge to a stable state. The aim of the Hopfield network is to recognize a partial or distorted input pattern as one of its previously memorized vectors and to output the perfect and complete memorized version.

Thus the Hopfield networks proposed as a theory of memory has the following features:

1. **Distributed Representation:** A memory is stored as a pattern of activation across a set of processing elements. Further, the memories can be superimposed on one another; different memories are represented by different patterns over the same set of processing elements.
2. **Distributed Asynchronous Controls:** Each processing element makes decisions based only on its own local situation. All these local actions add up to a global solution.
3. **Content-addressable Memory:** A number of patterns can be stored in a network. To retrieve a pattern, we need to only specify a portion of it. The network automatically finds the closest match.
4. **Fault Tolerance:** If a few processing elements misbehave or fails completely, the network will still function properly.

The main concept underlying the Hopfield network is that a single network of interconnected, binary-valued neurons can store multiple stable states, the library patterns also called *attractors*, similar to the brain – the full pattern can be recovered if the network is presented with only partial information just as described in associative memories. Furthermore, there is a degree of stability in the system, if just a few of the connections between neurons are severed, the recalled memory is not too badly corrupted, the network can respond with a "best guess". The nodes in the Hopfield network are vast simplifications of real neurons, they can only exist in one of two possible **"states"** - {firing, not firing} or $\{0, 1\}$ or $\{1, -1\}$. Every node is connected to every other node with some strength (synaptic weights). At any instant of time a node will change its state (i.e. start or stop firing) depending on the inputs it receives from the other nodes.

If we start the system off with any general pattern of firing and non-firing nodes then this pattern will in general change with time. To see this think of starting the network with just one firing node. This will send a signal to all the other nodes via its connections so that a short time later some of these other nodes will fire. These new firing nodes will then excite others after a further short time interval and a whole cascade of different firing patterns will occur. One might imagine that the firing pattern of the network would change in a complicated perhaps random way with time. The crucial property of the Hopfield network which renders it useful for simulating memory recall is the following: it guarantees that the pattern will settle down after a long enough time to some fixed pattern. Certain nodes will be always **"on"** and others **"off"**. Furthermore, it is possible to arrange that these stable firing patterns of the network correspond to the desired memories we wish to store.

Example 1: Suppose we have trained our Hopfield Net on the three patterns given in fig. 5:

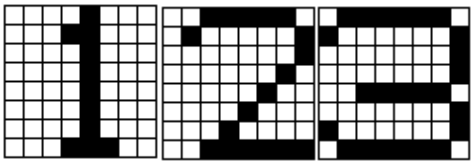


Fig. 5

Next we input a pattern which is a bit like one of these, say the left most one given in the Fig. 6. Leave the network to run. It gradually alters the pattern we give it until it has reconstructed one of the originals ones, the right most one.

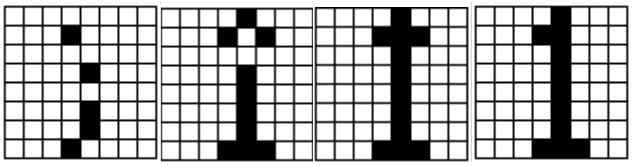


Fig. 6

The Hopfield Net is left to iterate (loop round and round) until it doesn't change any more. Then it *should* match one of the input patterns. The program should then compare the reconstructed pattern with the library of originals, and see which one matches.

9.4 STRUCTURE OF HOPFIELD NETWORKS

The structure of the Hopfield networks model is shown in Fig. 7.

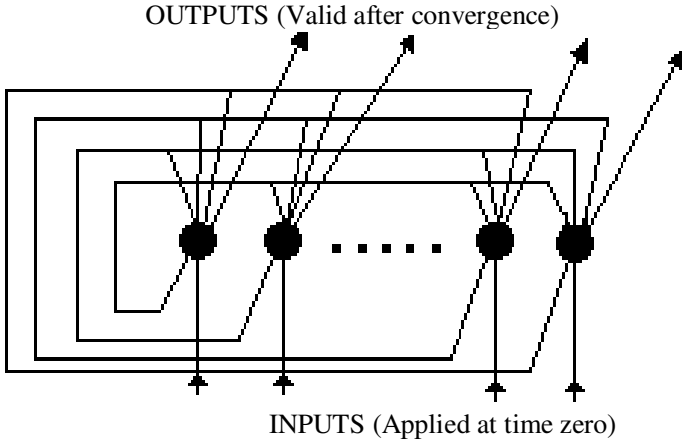


Fig. 7: Hopfield Network Model

The inputs to the Hopfield Net are at the bottom. The nodes produce an output which comes out at bottom and is fed back into all the nodes except the one that produced it (i.e. all the nodes refer to each other, but not to themselves) and uses this to produce the next output. The final output of the nodes is extracted at the top. For a network of N neurons, the state of the network is denoted by the vector,

$$\mathbf{s} = (s_1, s_2, \ldots, s_N)^T$$

Each $s_j = \pm 1$, the state of neuron j represents the one bit of information, and the $N \times 1$ state vector $\mathbf{s}$ represents a binary word of N bits of information. As the units/neurons

have a binary threshold units, the dynamical rule has the possibility of two definitions for a_i the activation of neuron i:

$$a_i = \begin{cases} 1 & \text{if } \sum_j w_{ij} s_j \geq \theta_i, \\ -1 & \text{otherwise.} \end{cases} \quad (4)$$

or,

$$a_i = \begin{cases} 1 & \text{if } \sum_j w_{ij} s_j \geq \theta_i, \\ 0 & \text{otherwise.} \end{cases} \quad (5)$$

where, w_{ij} is the connection (synaptic) weight from neuron i to neuron j as in Eqn. (1), s_j is the state of neuron (unit) j and θ_i is the threshold of the neuron i. The above relation in Eqns. (4) and (5) may be written in the compact form,

$$a_i = \text{sgn}[\Sigma w_{ij} s_j] \text{ for all } j = 1, 2, \ldots, N$$

where, sgn is the signum function. If a_i is zero then the action can be arbitrary. However, the neuron I may remain in the previous state regardless of whether it is on or off. The feedback network with no self-looping, Hopfield networks also have the following two restrictions on due to the kinds of interconnection that prevails:

i) $w_{ii} = 0$ for all neurons i. i.e. there is no loop/connection to itself, and

ii) $w_{ij} = w_{ji}$ for all neurons. i.e. the connections are symmetric.

The Hopfield nets also have a scalar value associated with each state of the network referred to as the energy, E of the network. Energy is defined as below,

$$E = -\sum_{i<j} w_{ij} s_i s_j + \sum_i \theta_i s_i \quad (6)$$

The reason for this is somewhat technical but we can proceed by analogy. Imagine a ball rolling on some bumpy surface. We imagine the position of the ball at any instant to represent the activity of the nodes in the network. Memories will be represented by special patterns of node activity corresponding to wells in the surface. Thus, if the ball is let go, it will execute some complicated motion but we are certain that eventually it will end up in one of the wells of the surface. We can think of the height of the surface as representing the energy of the ball. We know that the ball will seek to minimize its energy by seeking out the lowest spots on the surface-the wells.

In the language of memory recall, if we start/ initiate the network with a pattern for firing, the network must approximate one of the "stable firing patterns" in the memories. However, it will "under its own steam" end up in a nearby (with respect to the threshold) well in the energy surface thereby recalling the original perfect memory.

In the following section, we shall focus on the functionality of Hopfield networks.

9.5 THE FUNCTIONALITY OF HOPFIELD NETWORKS

As mentioned earlier that Hopfield nets are a special case of associative networks or content-addressable memory nets. Therefore, Hopfield nets too have the corresponding two phases namely, Storage phase and the Retrieval phase.

Storage Phase: Let us have M, N-dimensional vectors to be stored in the Hopfield nets, say $P = \{p_k \mid k = 1, 2, \ldots, M\}$. The m vectors compose the library or the fundamental memories, representing the patterns to be memorized by the network. Let $p_{k,I}$ denote the i-th element of the fundamental memory p_k. According to the Hebb's rule the synaptic weights from neuron i to j can be computed using Eqn. (1) i.e.

$$w_{ij} = 1/N(p_{1,i}\ p_{1,j} + p_{2,i}\ p_{2,j} + \cdots + p_{M,i}\ p_{M,j}) \tag{7}$$

and $w_{ii} = 0$ for all neurons $i = 1, 2, \ldots, M$. Let W denotes the $N \times N$ synaptic weight matrix or connectivity matrix of the network.

$$W = \frac{1}{N}\sum_{k}^{M} p_k p_k^T - MI \tag{8}$$

where, $p_k p_k{}^T$ denoted the dot product of the k-th vector and its transpose. I is the identity matrix. From the above computation the following three points are reconfirmed:

- The output of each neuron in the network is fed back to all other neurons.
- There is no self-looping in the network.
- The weight matrix of network is symmetric.

Retrieval Phase: This phase is initiated on input of an N-dimensional vector as input say $\mathbf{p}_t$, to the trained network. This is also called a probe. Typically the probe has elements ± 1 and is thought to be a noisy or incomplete version of any of the attractors already in the library/ fundamental memory. The information retrieval starts with a dynamical rule in which each neuron i of the network randomly but at some fixed rate examines its activation function a_i as a result of the connectivity its neighbouring neurons. While examining if a_i is greater than zero then the state is changed to 1 else remain in the previous state. But if the value of a_i is less than zero then switch state to -1. However, if the value of a_i is equal to zero then the state is left in its previous state regardless of whether it was in on or off state. The updating of the states of the neurons is deterministic but selection of the neurons for updating states is done randomly in serial (asynchronous) or synchronous. Starting with the test input or the probe vector the network must finally reach a state of stability, or produce time invariant state vector as the output pattern i.e. $\mathbf{y}$, whose individual elements satisfy the following stability condition,

$$y_i = \text{sgn}\left(\sum_{i=1}^{N} w_{j,i} p_{ti}\right),\ j = 1, 2, \ldots, N. \tag{9}$$

or the matrix form,

$$\mathbf{y} = \text{sgn}(\mathbf{W}\mathbf{p}_\mathbf{t}) \tag{10}$$

Example 2: Consider the following example:

i) Input Patterns $M = 3,\ N = 4,$ 3 4-dimensional vector.

IP1	1	1	1	–1
IP2	1	1	–1	1
IP3	– 1	1	1	–1

ii) The average correlation across the three patterns to design and train the Hopfield network.

$w_{i,j}$	IP1	IP2	IP3	Avg
$w_{1,2}$	1	1	–1	1/3
$w_{1,3}$	1	–1	–1	–1/3
$w_{1,4}$	–1	1	1	1/3
$w_{2,3}$	1	–1	1	1/3
$w_{2,4}$	–1	1	–1	–1/3
$w_{3,4}$	–1	–1	–1	–1

$$X = \begin{bmatrix} 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 \\ -1 & 1 & 1 & -1 \end{bmatrix}$$

$$X^T = \begin{bmatrix} 1 & 1 & -1 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ -1 & 1 & -1 \end{bmatrix}$$

The weight for the auto association network is $W = X^T X = \begin{bmatrix} 3 & 1 & -1 & 1 \\ 1 & 3 & 1 & -1 \\ -1 & 1 & 3 & -3 \\ 1 & -1 & -3 & 3 \end{bmatrix}$

Here it is clear that in the connectivity matrix $w_{2,4} = w_{4,2}$ or in general the lower and the upper triangles of the product matrix represents the 6 weights $w_{i,j} = w_{j,i}$. We can scale the weights by dividing them by $p = 3$. For the iterative purposes it is easier to scale by p at the end instead of scaling the entire weight matrix W prior to testing. Now let us construct the Hopfield net and training it with the inputs IP_1, IP_2, IP_3.

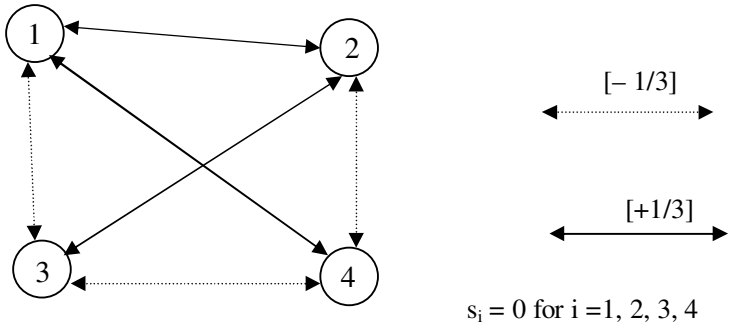


Fig. 8: Hopfield network

iii) Now consider the testing input = (1 0 0 –1).

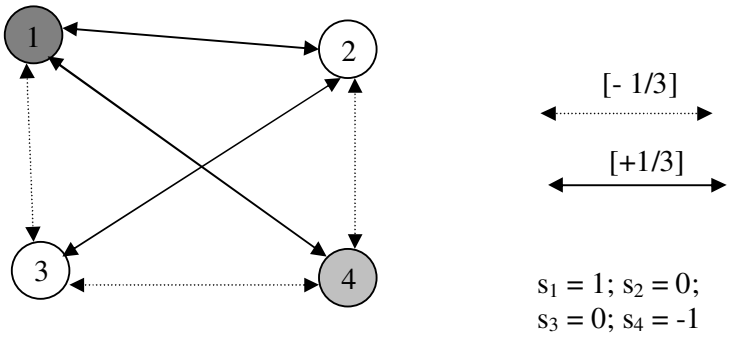


Fig. 9: With input (1 0 0 –1)

iv) Synchronous iteration to update all the nodes.

$$(1\ 0\ 0\ -1)\begin{pmatrix} 3 & 1 & -1 & 1 \\ 1 & 3 & 1 & -1 \\ -1 & 1 & 3 & -3 \\ 1 & -1 & -3 & 3 \end{pmatrix} = (2\ 2\ 2\ -2)$$

Table 1

Inputs					
Node	1	2	3	4	Output
1	**1**	0	0	-1/3	1
2	1/3	**0**	0	1/3	1
3	-1/3	0	**0**	1	1
4	1/3	0	0	**-1**	-1

The diagonal represents the values from the input layer in the above table depicting the result of the synchronous iteration modeled by the preceding computation.

Now scaling the output by dividing by $p = 3$ and using the threshold $\theta = 0$, we obtain the vector $(2/3\ 2/3\ 2/3\ -2/3) \Rightarrow (1\ 1\ 1\ -1)$ the same as IP_1 an attractor in the library. Consider the following figure to display the processing in the Hopfield net to reach a stable state again with the output vector as computed earlier. The graphical representation of the output is given in Fig. 10.

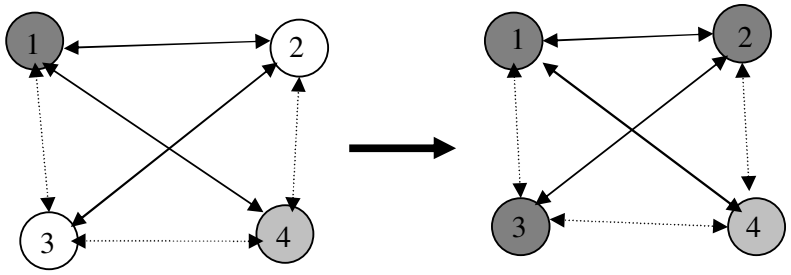


Fig. 10: (1 0 0 –1) → (1 1 1 –1)

Now, try some exercises.

E1) Run the above Hopfield network for the test input vector $(1\ 0\ 0\ 0)$.

E2) Draw the graph of the input to output vector obtained in E1).

The smart thing about the Hopfield network is that there exists a rather simple way of setting up the connections between nodes in such a way that any desired set of patterns can be made "stable firing patterns". Thus any set of memories can be burned into the network at the beginning. Then if we kick the network off with any old set of node activity we are *guaranteed* that a "memory" will be recalled. Not too surprisingly, the memory that is recalled is the one which is "closest" to the starting pattern. In other words, we can give the network a corrupted image or memory and the network will "all by itself" try to reconstruct the perfect image. Of course, if the input image is sufficiently poor, it may recall the incorrect memory–the network can become "confused"–just like the human brain. We know that when we try to remember someone's telephone number we will sometimes produce the wrong one! Notice also that the network is reasonably robust–if we change a few connection strengths just a little the recalled images are "roughly right". We don't lose any of the images completely.

Example 3: Consider a Hopfield network whose weight matrix is given by,

$$w = \frac{1}{3}\begin{bmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{bmatrix}$$

Consider a test input vectors $pt_1 = (1\ -1\ 1)$ and $pt_2 = (-1\ 1\ -1)$.

Using the Eqn. (9) and then its transpose results in the desired vector.

Therefore, $$\mathbf{W pt_1^T} = \frac{1}{3}\begin{bmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} = \frac{1}{3}\begin{bmatrix} 4 \\ -4 \\ 4 \end{bmatrix}$$

$$\mathbf{y_1} = \text{Sgn}\ (\mathbf{W pt_1}) = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$$

Since the CM or the weight matrix is symmetric we can also directly get the vector as by the product of vector and the matrix, i.e. **ptW**

$$\mathbf{y_2} = \text{sgn}\,(\mathbf{pt_2 W}) = (-1\ 1\ -1)\frac{1}{3}\begin{bmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{bmatrix} = (-1\ 1\ -1)$$

Eqn. (10) is also known as **alignment condition**. The state vectors that satisfy the alignment condition are stable states. Therefore, the given patterns are stables state vector too.

If we consider the following vectors as probes for this example, $(-1\ -1\ 1)$, $(1, 1, 1)$ or $(1\ -1\ -1)$. The resulting output is $(1\ -1\ 1)$. Note that each of the probes had single error compared to the stored memory.

Similarly, consider another set of test patterns to input to the Hopfield network $(1\,1\,-1)$, $(-1\,-1\,-1)$ or $(-1\,1\,1)$. The resulting output vector in this case is computed to be $(-1\,1\,-1)$. In this case too the test patterns are noted to have single error.

Now, try the following exercises.

E3) Run the Hopfield network of Example 2 for the test input vector $(1\,1\,0\,0)$. Also, draw the graphical representation of the output.

E4) Consider the set of pattern vectors P. Obtain the connectivity matrix (CM) for the patterns in P (four patterns).

$$P = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

In the following Section, we shall discuss the storage capacity Hopfield Networks.

9.6 HOPFIELD NETWORKS: STORAGE CAPACITY

There are two major limitations of Hopfield networks. One is the tendency to local minima, which is good for association but bad for optimisation. The other limitation is on the network capacity. For a network of N binary nodes, the capacity limit is of the order of N rather than 2^N. If the patterns or the vectors are N-dimensional then there must be network of N binary nodes.

However, the capacity is equal to the relationship between the number of patterns that can be stored & retrieved without error to the size of the network. Let M be the number of patterns to be stored or fundamental memories. So long as the storage capacity is of the network is not overloaded i.e., M is small compared to N.

$$\text{Capacity} = \frac{M}{N} \text{ or, } \frac{M}{\text{Number of weights}}$$

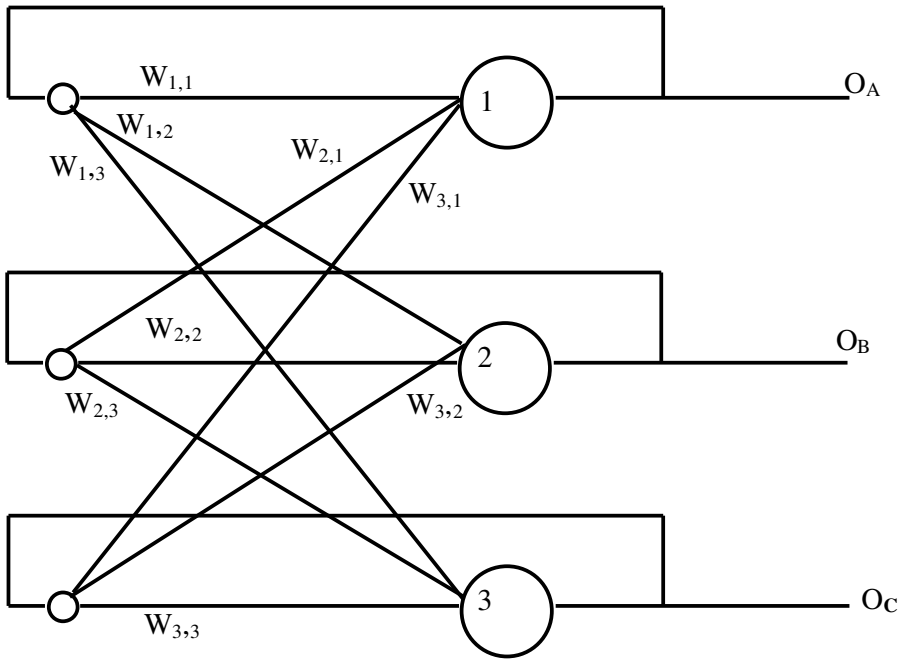
If we use the following definition of 100% correct retrieval, when any of the stored patterns is entered completely (no noise), then that same pattern is returned by the network; i.e. The pattern is a stable attractor. A detailed proof shows that a Hopfield network of N nodes can achieve 100% correct retrieval on P patterns if:
$P < N/(4 * \ln(N))$.

N	Max P
10	1
100	5
1000	36
10000	271
10^{11}	10^9

In general, as more patterns are added to a network, the average correlations will be less likely to match the correlations in any particular pattern. Hence, the likelihood of retrieval error will increase. Just as it happens in case of human memory. When the information is loaded/crammed into a memory, or patterns to store recall is slower

than when there are less patterns to remember. The key to perfect recall is selective ignorance!

Example 4: A three neuron network trained with three patterns.



Let's say we'd like to train three patterns:

Pattern number one: $O_{A(1)} = -1\ O_{B(1)} = -1\ O_{C(1)} = -1$

Pattern number two: $O_{A(2)} = 1\ O_{B(21)} = -1\ O_{C(2)} = -1$

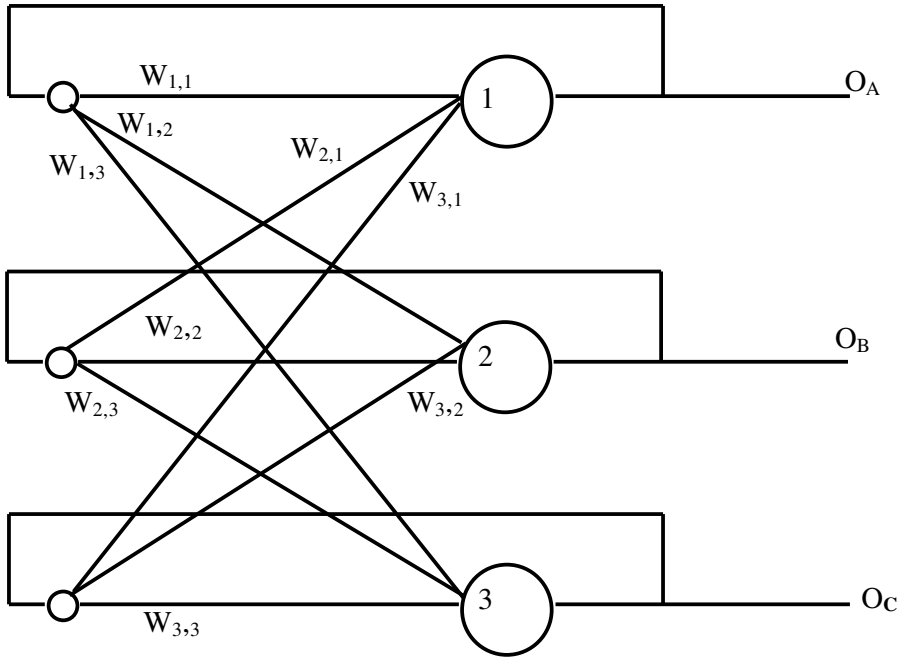
Pattern number three: $O_{A(3)} = -1\ O_{B(3)} = 1\ O_{C(3)} = 1$

$$W_{1,1} = 0$$
$$W_{1,2} = O_{A(1)} \times O_{B(1)} + O_{A(2)} \times O_{B(2)} + O_{A(3)} \times O_{B(3)} = (-1)\times(-1) + 1\times(-1) + (-1)\times 1 = -1$$
$$W_{1,3} = O_{A(1)} \times O_{C(1)} + O_{A(2)} \times O_{C(2)} + O_{A(3)} \times O_{C(3)} = (-1)\times 1 + 1\times(-1) + (-1)\times 1 = -3$$

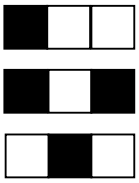
$$W_{2,2} = 0$$
$$W_{2,1} = O_{B(1)} \times O_{A(1)} + O_{B(2)} \times O_{A(2)} + O_{B(3)} \times O_{A(3)} = (-1)\times(-1) + (-1)\times 1 + 1\times(-1) = -1$$
$$W_{2,3} = O_{B(1)} \times O_{C(1)} + O_{B(2)} \times O_{C(2)} + O_{B(3)} \times O_{C(3)} = (-1)\times 1 + (-1)\times(-1) + 1\times 1 = 1$$

$$W_{3,3} = 0$$
$$W_{3,1} = O_{C(1)} \times O_{A(1)} + O_{C(2)} \times O_{A(2)} + O_{C(3)} \times O_{A(3)} = 1\times(-1) + (-1)\times 1 + 1\times(-1) = -3$$
$$W_{3,2} = O_{C(1)} \times O_{B(1)} + O_{C(2)} \times O_{B(2)} + O_{C(3)} \times O_{B(3)} = 1\times(-1) + (-1)\times(-1) + 1\times 1 = 1$$

E5) Train this network with the three patterns shown.



Patterns:



Applications of the Hopfield Network

1. The Hopfield network can be used as an effective interface between analog and digital devices, where the input signals to the network are analog and the output signals are discrete values.
2. Associative memory is a major application of the Hopfield network.
3. The energy minimization ability of the Hopfield network is used to solve optimization problems.
4. The various applications of Hopfield networks are as given below. Hopfield network remembers cues from the past and does not complicate the training procedure.
5. The Hopfield network can be used for converting analog signals into the digital format, for associative memory.

Now, let us summarize this unit.

9.7 SUMMARY

The summary of this unit includes the four basic operations to design and run a Hopfield network. These four operations are–learning, initialization (of the network), iteration until convergence, and finally outputting the output vector.

i) **Learning/Training :** Let $p_1, p_2, \ldots, p_m$ be the N-dimensional patterns to be stored. Using the dot product rule, the Hebbian postulate of learning, the synaptic weights of the entire recurrent network prohibiting self-looping is computed by,

$$w_{i,j} = \begin{cases} \frac{1}{N}\sum_{k=1}^{M} p_{k,i}p_{k,j} & i \neq j \\ 0 & i = j \end{cases}$$

ii) **Initialization:** Let $\mathbf{p}_t$ be the n-dimensional vector probe or the test input to the Hopfield network. The initialization is carried out by,

$$s_i(0) = p_{i,t},\ i = 1, 2, \ldots, N$$

where $s_i(0)$ is the state of the i-th neuron at time $n = 0$, and $p_{i,t}$ the i-th element of the test input p_t.

iii) **Iteration until convergence:** Update the elements of the state vector $s(n)$ synchronously or asynchronously using the rule,

$$s_i(n+1) = \text{sgn}\left[\sum_{j=1}^{N} w_{ij}s_i(n)\right],\ i = 1, 2, \ldots, N$$

Repeat the iteration until the state of the neurons do not change i.e. the state vector s remains unchanged.

iv) **Outputting:** Let s_{fixed} denote the stable states computed at the end of above iterative step. The resulting pattern OP of the network is,

$$OP = s_{fixed}$$

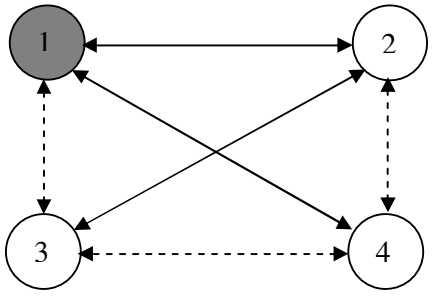
The storage phase uses the learning/training operation. The subsequent operations initialization, iteration until convergence and outputting are applied in the retrieval phase.

9.8 SOLUTIONS/ANSWERS

E1) This test pattern too has elements other than ± 1. Using the Eqn. (4) or **ptW** and dividing the resultant by 3 the vector obtained is $(1\ 1\ -1\ 1)$. The calculation is given as below.

Input = (1 0 0 0)

The corresponding graphical representation of the Hopfield network is given as



The synchronous iteration to update the nodes is given by

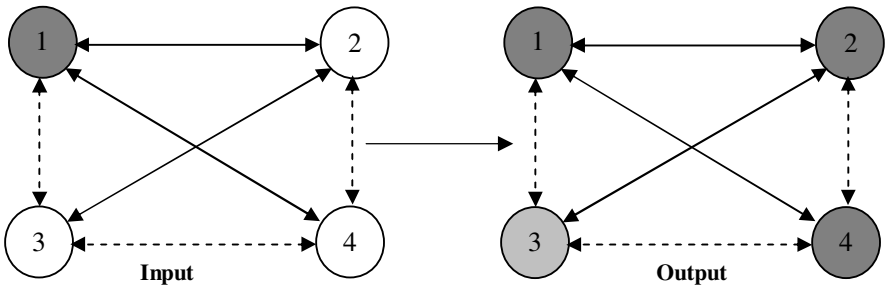
$$(1 \quad 0 \quad 0 \quad 0)\begin{pmatrix} 3 & 1 & -1 & 1 \\ 1 & 3 & 1 & -1 \\ -1 & 1 & 3 & -3 \\ 1 & -1 & -3 & 3 \end{pmatrix}$$

$$= (3 \quad 1 \quad -1 \quad 1)$$

Nodes	1	2	3	4	Output
1	**1**	0	0	0	1
2	1/3	**0**	0	0	1
3	– 1/3	0	**0**	0	-1
4	1/3	0	0	**0**	1

Which is a stable state and one of the input patterns stored in the library. Though the asynchronous update results in spurious output, the synchronous update gives a pattern from the fundamental memory.

E2)



E3) First of all the input definitely has some serious noise therefore the value 0 which is not there in the input patterns is present in the probe. However, use the Eqn. (4) or $\mathbf{ptW} = (1\ 1\ 0\ 0)$, which is a stable state, but not one of the input patterns stored in the library.

E4) Here $N = 10$ and $M = 4$. Use either the Eqn. (6) or Eqn. (7) to obtain the weights for each $w_{i,j}$ for all i and $j = 1, 2, \ldots, 10$ but $i \neq j$. Substitute the values of P^T and P in Eqn. (7) we get $CM = 1/4P^TP - 10I_{10}$.

9.9 PRACTICAL ASSIGNMENT

Session 8

Write a program in 'C' language to find

i) The average correlation matrix for given input patterns M and N to design and train the Hopfield Network.

ii) The weight matrix.

iii) Output of the Hopfield network.

Also test your program on the input patterns given in Example 2.

UNIT 10 KOHONEN NETWORKS

Structure **Page No.**

10.1 INTRODUCTION

Typical neural network tries to simulate the biological human brain. Explaining how the human brain learns is quite difficult for a simple reason: nobody knows it exactly. However, it is known that human brain consists of a large number of neural cells that process information. Each cell works like a simple processor and only the massive interaction between all cells and their parallel processing makes the brain's abilities possible. If a certain amount of electrical stimulation is received by a neuron, it generates an output to all other connected neurons. Now if the incoming stimulation is too low, no output is generated by the neuron and the information's further transport will be blocked. On the other hand, if a certain amount of stimulation is received by a neuron, it generates an output to all other connected neurons and so information takes its way to its destination where some reaction will occur. It is supposed that during the learning process the connection structure among the neurons is changed, so that certain stimulations are only accepted by certain neurons. This means that the firm connections exist between the neural cells that once have learned a specific fact, enabling the fast recall of this information. Unlike the biological model, a neural net has an unchangeable structure. It is built of a specified number of neurons and a specified number of connections between them–*weights*–which have certain values.

The Kohonen network is named after the Finnish researcher, Teuvo Kohonen from University of Helsinki who pioneered the research and first presented in 1982. A Kohonen network is a two-layered network, much like the Perceptron. But the output layer for a two-neurone input layer can be represented as a two-dimensional grid, also known as the "competitive layer", which we shall discuss in Sec. 10.2.

To the supervised learning algorithms the desired outputs are provided for the comparison with the actual outputs in order to measure performance of the systems or the error caused by the system. The difference between actual output and target, an error value is also used for adjusting the weights between neurons. It is known that the *cortex* of the human brain is subdivided in different regions, each responsible for certain functions. To simulate this model of human brain, neural cells are geared to organize themselves in groups, according to incoming information. The Kohonen network is most famous neural network model designed in accordance to this type of *unsupervised learning*. Here the desired output is not provided during the modeling phase, therefore Kohonen Networks function based on the principles of *competitive learning* or *self-organization*. We shall discuss Kohonen networks in Secs. 10.3, 10.4 and 10.5 and self organizing feature map in Sec.10.6.

Objectives

After reading this unit will be able to:

- clarify about different types Artificial Neural Networks;
- differentiate Kohonen networks from other feed forward back propagation neural networks;
- define and explain the fundamentals, the types, the basic algorithm of Kohonen networks;
- explain the terms unsupervised learning, competitive learning, SOM, etc;
- apply the Kohonen Networks, SOM for the kinds of learning problems they were designed.

10.2 COMPETITIVE LEARNING

Competitive learning: *When one student achieves the goal all other students fail to reach that goal.*

A basic **competitive learning** network has one layer of input neurons and one layer of output neurons. An input pattern x is a sample point in the n-dimensional real or binary vector space. Binary-valued (1 or 0) *local representations* are more often used for the output nodes. That is, there are as many output neurons as the number of classes and each output node represents a pattern category.

A competitive learning network comprises the feed forward excitatory network(s) and the lateral inhibitory network(s). The feed forward network usually implements an excitatory **Hebbian learning rule**. It consists of when an input cell persistently participates in firing an output cell, the input cell's influence firing that output cell is increased. The lateral competitive network is inhibitory in nature. The network serves the important role of selecting the winner, often via a competitive learning process, highlighting the **"winner-take-all"** schema. In a winner-take-all circuit, the output unit receiving the largest input is assigned a full value (e.g. 1), whereas all other units are suppressed to a 0 value. The winner-take-all circuit is usually implemented by a (digital or analog) MAXNET network. Another example of a lateral network is Kohonen's self-organizing feature map. By allowing the output nodes to interact via the lateral network, the neural model can be trained to preserve certain topological ordering.

Using no supervision from any teacher, unsupervised networks adapt the weights and verify the results only on the input patterns. One popular scheme for such adaptation is the competitive learning rule, which allows the units to compete for the exclusive right to respond to a particular input pattern. It can be viewd as a sophisticated clustering technique, whose objective is to divide a set of input patterns into a number of clusters such that the patterns of the same cluster exhibit a certain degree of similarity. The training rules are often the Hebbian rule for the feed forward network and the winner-take-all (WTA) rule for the lateral network.

Goals of Competitive Learning

1. **Error Minimization** – Minimizing the quantization (distortion) errors. A typical application where error minimization is important is *vector quantization*. In vector quantization data is transmitted over limited bandwidth communication channels by transmitting for each data vector only the *index* of the nearest reference vector. The set of reference vectors (which is called *codebook* in this context) is assumed to be known both to sender and receiver. Therefore, the receiver can use the transmitted indexes to retrieve the

corresponding reference vector. There is an information loss in this case which is equal to the distance of current data vector and nearest reference vector. The expectation value of this error is described by equations and in particular if the data distribution is clustered (contains sub regions of high probability density), dramatic compression rates can be achieved with vector quantization with relatively little distortion.

2. **Entropy maximization** – Sometimes the reference vectors should be distributed such that each reference vector has the same chance to be winner for a randomly generated input signal ξ:

$$P(s(\xi) = c) = \frac{1}{|A|} \quad (\forall c \in A)$$

An advantage of choosing reference vectors such as to maximize entropy is the inherent robustness of the resulting system. The removal (or ``failure'') of any reference vector affects only a limited fraction of the data.

3. **Feature Mapping** – With some network architectures it is possible to map high-dimensional input signals onto a lower-dimensional structure in such a way, that some similarity relations present in the original data are still present after the mapping. This is denoted by *feature mapping* and can be useful for data visualization. A prerequisite for this is that the network used has a fixed dimensionality. This is the case, e.g., for the *self-organizing feature map* and the other methods discussed in section of this report. A related question is, how *topology-preserving* is the mapping from the input data space onto the discrete network structure, i.e., how well are similarities preserved? Several quantitative measures have been proposed to evaluate this like the topographic product or the topographic function.

4. **Other** – Clustering, density estimation, combining with supervised learning (RBF) obtain better results.

Now, try the following exercise.

E1) Write any two goals of competitive learning.

Now, let us begin the following section by defining the Kohonen networks.

10.3 KOHONEN NETWORKS

The objective of a Kohonen network is to map input vectors (patterns) of arbitrary dimension N onto a discrete map with 1 or 2 dimensions. Patterns close to one another in the input space should be close to one another in the map: they should be topologically ordered. A Kohonen network is composed of a grid of output units and N input units. The input pattern is fed to each output unit. The input lines to each output unit are weighted. These weights are initialized to small random numbers.

Distinctiveness of Kohonen Networks

The Kohonen neural network differs from the usual feed forward back propagation neural network not in architecture but in its functionality. The Kohonen neural network is different both in how it is trained and how it recalls a pattern. The Kohonen neural network does not use any activation function and also a bias weight. The output from the Kohonen neural network is not a composition of the output of several neurons. On input of a pattern to a Kohonen network one of the output neurons is selected as a "winner". This "winning" neuron is the output from the

Kohonen network. Often these "winning" neurons represent groups in the data that is presented to the Kohonen network.

The most significant difference between the Kohonen neural network and the feed forward back propagation neural network is that the Kohonen network trained in an unsupervised mode. This means that the Kohonen network is presented with data, but the correct output that corresponds to that data is not specified. Using the Kohonen network this data can be classified into groups. We will begin our review of the Kohonen network by examining the training process.

It is also important to understand the limitations of the Kohonen neural network. You will recall from the previous unit that neural networks with only two layers can only be applied to linearly separable problems. This is the case with the Kohonen neural network. Kohonen neural networks are used because they are a relatively simple network to construct that can be trained very rapidly.

Basic Algorithm

The algorithm for training a Kohonen network can be summarized in the following steps:

Step 1: Apply an input vector X to the network.

Step 2: Calculate the distance D_j (in n-dimensional space) between X and the weight vectors W_j of each neuron.

Step 3: The neuron that has the weight vector closest to X is declared the winner. Use this weight vector W_c as the centre of a group of weight vectors that lie within a distance of d from W_c.

Step 4: Train this group of vectors according to for all weight vectors within a distance d of W_c.

Step 5: Perform steps 1 through 4 for each input vector.

The n connection weights into a neuron are treated as a vector in n-dimensional space. Before training, the vector is initialized with random values, and then the values are normalized to make the vector of unit length in weight space. The input vectors in the training set are likewise normalized. As training proceeds, the values of d are gradually reduced. It is recommended by Kohonen that start near 1 and reduce to 0.1, whereas d can start as large as the greatest distance between neurons and reduce to a single neuron.

Now try the following exercise.

E2) Differentiate Kohonen networks for the feed forward and back propagation neural network.

Now, in the following section, we shall discuss the structure of Kohonen networks in detail.

10.4 STRUCTURE OF KOHONEN NETWORKS

The Kohonen neural network works differently than the feed forward neural network. The Kohonen neural network contains only an input and output layer of neurons.

There is no hidden layer in a Kohonen neural network. First we will examine the input and output to a Kohonen neural network.

The input to a Kohonen neural network is given to the neural network using the input neurons. These input neurons are each given the floating point numbers that make up the input pattern to the network. A Kohonen neural network requires that these inputs be normalized to the range between -1 and 1. Presenting an input pattern to the network will cause a reaction from the output neurons.

The output of a Kohonen neural network is very different from the output of a feed forward neural network. In a typical feed-forward network with five output neurons the number of output values is consisted to be five. However, in the case of a Kohonen neural network only one of the five output neurons actually produces a value. Additionally, this single value is either true or false. When the pattern is presented to the Kohonen neural network, one single output neuron is chosen as the output neuron. Therefore, the output from the Kohonen neural network is usually the index of the neuron (i.e. Neuron $\#5$) that fired. The structure of a typical Kohonen neural network is shown in Fig. 1.

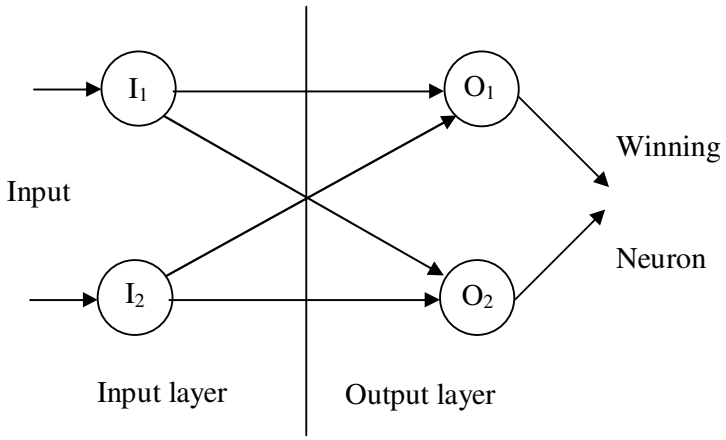


Fig. 1: A Kohonen Neural Network

Example 1: Consider a very simple Kohonen neural network. This network includes only two input and two output neurons. The input given to the two input neurons is shown in Table 10.1.

Table 10.1: Inputs to a Kohonen Neural Network

Input Neuron 1 (I_1)	0.5
Input Neuron 2 (I_2)	0.75

Consider the connection weights between the neurons as given in Table 10.2.

Table 10.2: Connection weights in the sample Kohonen neural network

$I_1 -> O_1$	0.1
$I_2 -> O_1$	0.2
$I_1 -> O_2$	0.3
$I_2 -> O_2$	0.4

Using these values we examine which neuron would win and produce output. This can be achieved by normalizing the input.

Normalization of Inputs

The Kohonen neural network also requires that its input be normalized. Kohonen neural network is considered to have a two layer network because there are only two actual neuron layers at work in the Kohonen neural network.

The requirements that the Kohonen neural network places on its input data are one of the most sever limitations of the Kohonen neural network. Input to the Kohonen neural network should be between the values -1 and 1. In addition, each of the inputs should fully use the range. If one, or more, of the input neurons were to use only the numbers between 0 and 1, the performance of the neural network would suffer.
To normalize the input we must first calculate the "vector length" of the input data, or vector. This is done by summing the squares of the input vector.

Example 2: Normalize the input given in Example 1.

Solution: Normalization of input gives us $= (0.5*0.5)+(0.75*0.75)$
This would result in a "vector length" of 0.8125. If the length becomes too small, say less than the length is set to that same arbitrarily small value. In this case the "vector length" is a sufficiently large number. Using this length we can now determine the normalization factor. The normalization factor is the reciprocal of the square root of the length. For our value the normalization factor is calculated as follows.

$$\frac{1}{\sqrt{0.8125}}.$$

This results in a normalization factor of 1.1094. This normalization process will be used in the next step where the output layer is calculated.

Calculating Each Neuron's Output

To calculate the output the input vector and neuron connection weights must both be considered. First the "dot product" of the input neurons and their connection weights must be calculated. To calculate the dot product between two vectors you must multiply each of the elements in the two vectors. We will now examine how this is done.

The Kohonen algorithm specifies that we must take the dot product of the input vector and the weights between the input neurons and the output neurons.

Example 3: Calculate the output of each of the neuron of Example 1.

Solution: The output from the neuron 1 is

$$|0.5\ \ 0.75|\bullet|0.1\ \ 0.2| = (0.5*0.75)+(0.1*0.2)\ = 0.395$$

The output from the neuron 2 is $|0.5\ \ 0.75|\bullet|0.3\ \ 0.4| = 0.495$

In the following section, we shall discuss the methods for modifying the weights.

10.5 METHODS FOR WEIGHT UPDATES

There are several methods for updating a neuron's weight vector in response to input pattern. The original algorithm proposed by Kohonen is to add a fraction of the input

vector to the weight vector, renormalizing the sum vector after that. This pushes the weight vector in the direction of the data vector "bit by bit". If x is the input vector presented to the network, and w^t presents the weight vector of the winning neuron at time t, then the updated weight vector w^{t+1} can be calculated as,

$$w^{t+1} = w^t = \alpha x / | w^t + \alpha x |$$

Denominator in this equation calculates the length (Euclidean norm) of the vector, and the constant α is called the *learning rate*. It is always much less than 1, and is usually decreased as training progresses.

Another, subtractive method relies on the fact that a weight vector can be pushed toward input data vector by subtracting them (thus finding the error difference), and adding the fraction of this difference to the weight vector.

$$\varepsilon = x - w^t$$

$$w^{t+1} = w^t + \alpha\varepsilon$$

There are many fine details that should be studied in order to successfully use this type of neural network in your projects. These include *conscience* mechanism, *inhibition* algorithms, etc. Below are some excellent sites that you can use to gain more knowledge on this subject.

Error Calculation

Unlike the case of supervised learning where the anticipated output of the neural network before training the network is available for comparing it with the actual output. Therefore the error is computed as the difference between the anticipated output of the neural network and the actual output of the neural network. However, in case of unsupervised training there is no anticipated output. Therefore the calculated error is not the true error, or at least not an error according to the normal understanding.

The purpose of the Kohonen neural network is to classify the input into several sets. The error for the Kohonen neural network, therefore, must be able to measure how well the network is classifying these items. Two methods for determining the error in are examined in this section. There is no official way to calculate the error for a Kohonen neural network. The error is just a percent number that gives an idea of how well the Kohonen network is classifying the input into the output groups. The error itself is not used to modify the weights, as was done in the back propagation algorithm.

Now, let us discuss the self-organizing feature map in the following section.

10.6 SELF-ORGANIZING FEATURE MAP

Kohonen networks are based on the concept of competitive learning. However, based on their application for various purposes Kohonen networks are further categorized in two types: Self-organizing Feature Maps and Vector Quantization. Here we elaborate upon the most popularly used Kohonen model i.e. Self-Organizing Maps (SOM).

Self-organizing system is a special class of ANN based on competitive learning. The input layer contains n neurons to input the pattern while in the output layer the neurons of the network compete among themselves to be activated or fired with the result that only one output neuron is on at any one time. The output neuron that wins the competition is called a winner-takes-all or simply the winning neuron. One way of inducing a winner-take-all competition is to use the lateral inhibitory connections i.e. negative feed-back paths between them.

In a self-organizing map, the neurons are placed at the nodes of a lattice that is whether one- or two-dimensional. The neurons become tuned to various input patterns or the classes of the input patterns in the course of a competitive learning process. The location of the winning neuron becomes ordered with respect to each other in such a way that a meaningful coordinate system for different input features is created over the lattice. A self-organizing map is therefore characterized by the formation of a topographic map of the input patterns in which the coordinates (spatial locations) of the neurons in the lattice are indicative of the statistical features contained in the input patterns. The self-organization maps are inherently non-linear. This class of neural network model is motivated by the features of the human brain. Human brain is organized in many places in such a way that different sensory inputs are represented by topologically ordered computational maps. For example, sensory inputs such as visual, acoustic, tactile, etc. are mapped onto different areas of the cerebral cortex in a topologically ordered manner. A computational map is defined by an array of neurons representing slightly differently tuned processors which operate on the sensory information bearing signals in parallel. Thus, the neurons are expected to transform input signals into a place coded probability distribution that represents the computed values of parameters by sites of relative activity within the map.

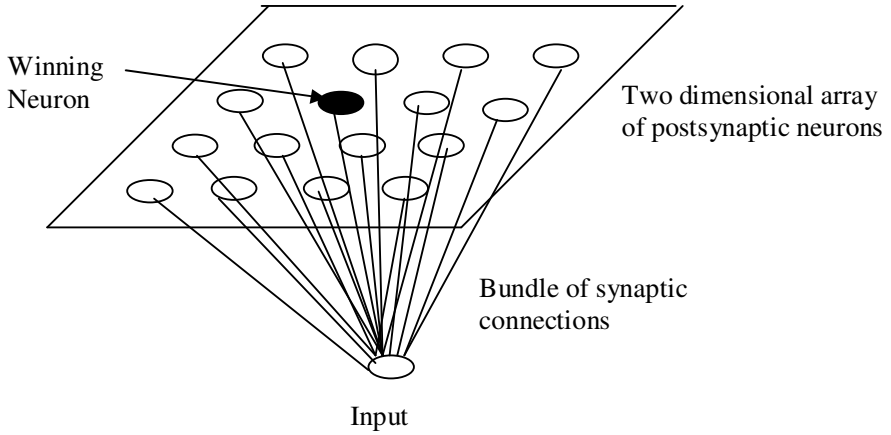


Fig. 2: Basic model of SOM

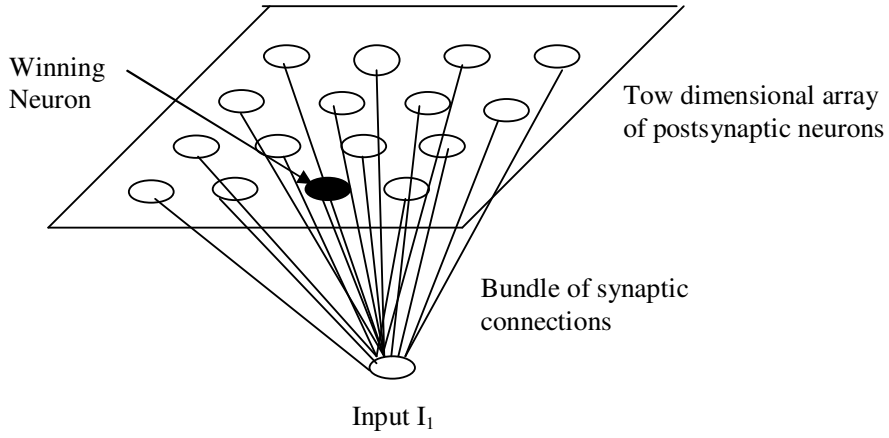


Fig. 3: SOM for an input pattern I_1

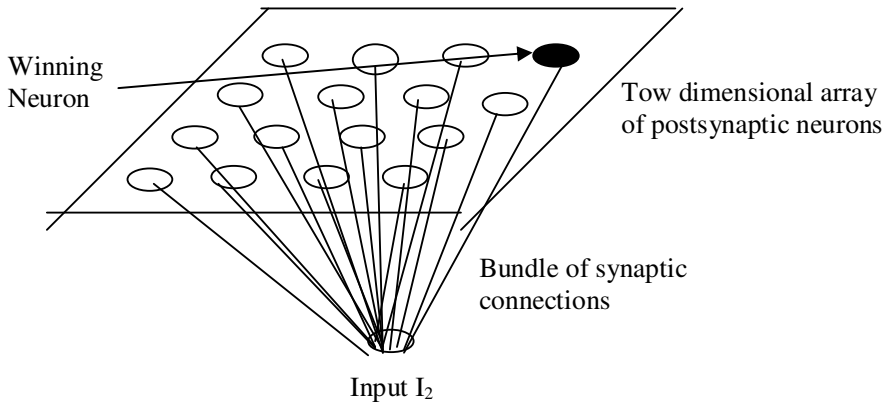


Fig. 4: SOM for an input pattern I_2

Fig. 2 presents the basic models for feature mapping as proposed by Kohonen while Fig. 3 and Fig. 4 exhibit the winning neuron to be different in case the input patterns I_1 and I_2 are not similar. The use of computational maps offers the following properties:

1. At each stage of representation, each incoming piece of information is kept in its proper context,
2. Neurons dealing with closely related pieces of information are close together so that they can interact via short synaptic connections.

From the design of self-organizing maps using computational maps the following principle of topographic map formulation was proposed by Kohonen:

The spatial location of an output neuron in a topographic map corresponds to a particular domain or feature of data drawn from the input space.

The principle goal of the SOM is to transfer an incoming signal pattern to arbitrary dimension into a one- or two-dimensional discrete map and to do this transformation adaptively in a topologically ordered fashion. The above model of SOM captures the essential features of computational maps in the brain and yet remains computationally tractable. This model also belongs to the class of vector-coding algorithms. It provides a topological mapping that optionally places a fixed number of vectors (code words) into a higher dimensional input space and thereby facilitates data compression.

The algorithm for the training of the self-organizing maps has two fundamental steps. The first step is for the initializing the synaptic weight of the network and the second step is for the training. Initialization is done by assigning small values picked from a random number generator. For doing this no prior order is imposed on the feature map. After initialization, in the training step the following three processes are executed:

1. Competition,
2. Cooperation, and
3. Synaptic Adaptation

Competition: For each input pattern, the neuron in the network compute their respective values of a discriminate function. This discriminate function provides the basis for competition among the neurons. The neuron with the largest value of discriminate function is declared winner of the competition.

Cooperation: The winning neuron determines the spatial location of a topological neighbourhood of excited neurons. Thus, it provides the basis for cooperation among the neighbouring neurons.

Synaptic Adaptation: This process enables the excited neuron to increase its individual value of the discriminate function in relation to the input pattern through suitable adjustments applied to their synaptic weights. The adjustments made are influenced by the response of the winning neuron to a subsequent application of a similar input pattern in an enhanced manner.

Example 4 (A Graphic Example):

Here we illustrate the self-organisation of a Kohonen net graphically. Consider an input space with two components only. Let the lattice include 6 nodes on a rectangular grid where the winner neuron will be excited as output.

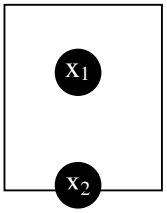


Fig. 5: Input neurons

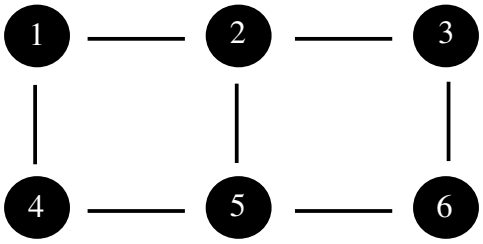


Fig. 6: Kohonen net of 6 neurons net on grid (output lattice)

Since the input space having only two components each of the neurons on the lattice will be initialized with randomly assigned/selected two weights. Thus another representation of the Kohonen net (lattice) may be in the *weight space* as presented in the Fig. 7. The lines denote the connection between the adjacent neurons in the initial output lattice as in Fig. 6.

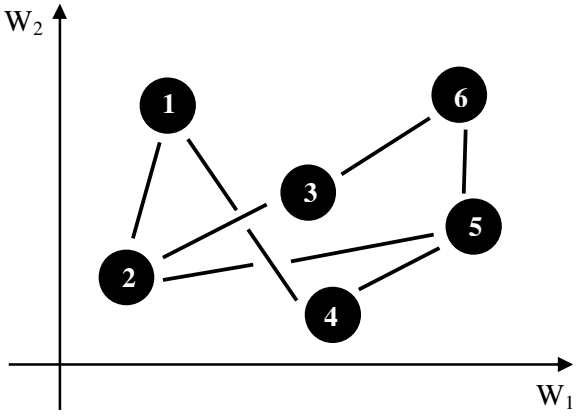


Fig. 7: Output lattice of Fig. 6 in weight space

Input Patterns: Suppose now that there are 6 input patterns (vectors). The representation of the patterns A, B, C, D, E and F corresponding the two components x_1 and x_2 is as shown below.

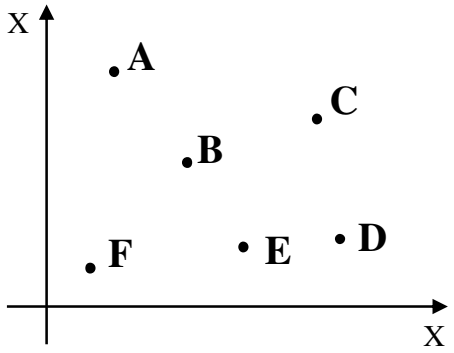


Fig. 8: Input patterns in two component space

In a well trained (ordered) net that has developed a topographic map the diagram in weight space should have the same topology as that in physical space and therefore should reflect the properties of the training set.

Trained weight space

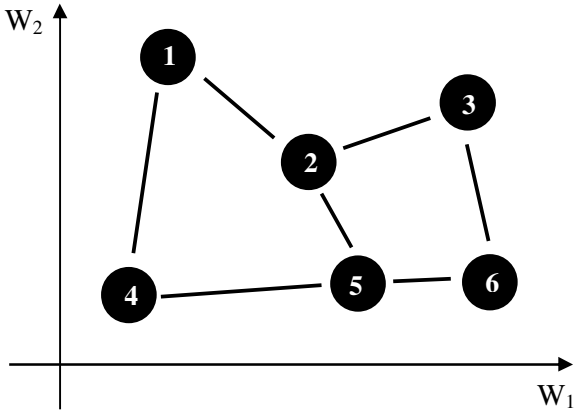
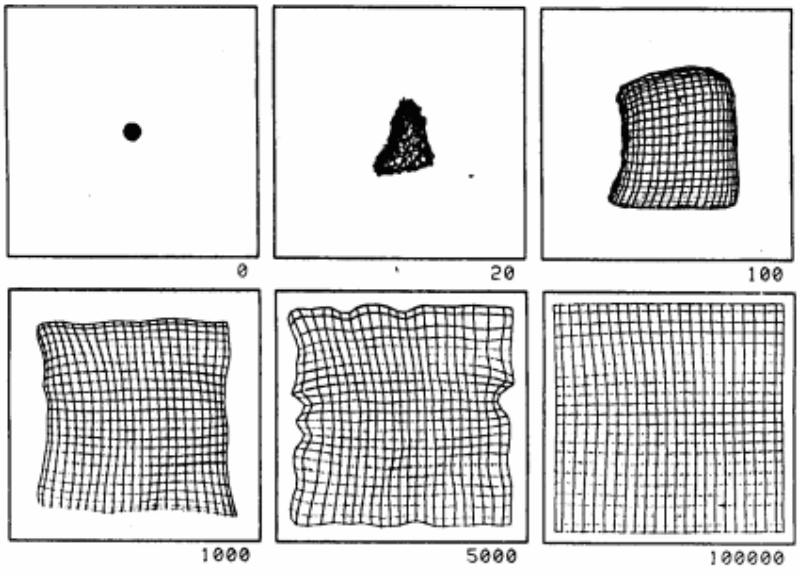


Fig. 9: Neurons trained based on the 6 input patterns A, B, C, D, E and F

Example 5: Self-Organizing Feature Maps



The case of 2-component vectors which are drawn randomly from the unit square and in which the initial weights are close to the centre of the unit square, is dealt with by Kohonen as in the above picture.

The weight diagram starts as a 'crumpled ball' at the centre and expands like fishing net being unraveled. Kohonen deals with several other similar examples.

When the input space is more than 2-dimensional, the higher dimensions have to get 'squashed' onto the grid. This will be done in such a way as to preserve the most important variations in the input data. Thus, it is often the case that the underlying dimensionality of the input space is smaller than the number of input patterns.

Example 6: If the input vector are $I_1 = [-1\ 0]^T, I_2 = [0\ 1]^T$ and $I_3 = [\sqrt{2}\ \ 1/\sqrt{2}]^T$ and initial values of three weight vectors are $[0\ -1]^T, [-2/\sqrt{5}\ \ 1/\sqrt{5}]^T$, $[-1/\sqrt{5}\ \ 2/\sqrt{5}]^T$, calculate the resulting weight found after training the competitive layer with Kohonen's rule and a learning rate α of 0.5 on the input-series in order I_1, I_2 and I_3.

Solution: First combining weight-vectors into a weight matrix

$$W = \begin{bmatrix} 0 & -1 \\ -2/\sqrt{5} & -1/\sqrt{5} \\ -1/\sqrt{5} & 2/\sqrt{5} \end{bmatrix}$$

Now presenting first-input vector I_1:

$$a = f(W.I_1) = f\left(\begin{bmatrix} 0 & -1 \\ -2/\sqrt{5} & 1/\sqrt{5} \\ 1/\sqrt{5} & 2/\sqrt{5} \end{bmatrix}\begin{Bmatrix} -1 \\ 0 \end{Bmatrix}\right) = f\left(\begin{bmatrix} 0 \\ 0.894 \\ 0.447 \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

(As the function f ( ) is competitive function.)
The second neuron responded, since $W^{(2)}$ is closest to I_1, so update $W^{(2)}$ with Kohonen's rule:

$$\begin{aligned} W_{new}^{(2)} &= W_{old}^{(2)} + \alpha\left(I_1 - W_{old}^{(2)}\right) \\ &= \begin{bmatrix} -2/\sqrt{5} \\ 1/\sqrt{5} \end{bmatrix} + 0.5\left(\begin{bmatrix} -1 \\ 0 \end{bmatrix} - \begin{bmatrix} -2/\sqrt{5} \\ 1/\sqrt{5} \end{bmatrix}\right) \\ &= \begin{bmatrix} -0.947 \\ 0.224 \end{bmatrix} \end{aligned}$$

Now present second input I_2:

$$a = f\left(W.I_2\right) = f\left(\begin{bmatrix} 0 & -1 \\ -0.947 & 0.224 \\ -1/\sqrt{5} & 2/\sqrt{5} \end{bmatrix}\begin{bmatrix} -1 \\ 0 \end{bmatrix}\right)$$

$$= f\left(\begin{bmatrix} -1 \\ 0.224 \\ 0.894 \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Here third neuron won, so its weights move closer to I_2 :

$$W_{new}^{(3)} = W_{old}^{(3)} + \alpha\left(I_1 - W_{old}^{(3)}\right)$$

$$= \begin{bmatrix} -2/\sqrt{5} \\ 1/\sqrt{5} \end{bmatrix} + 0.5\left(\begin{bmatrix} 0 \\ 1 \end{bmatrix} - \begin{bmatrix} -1/\sqrt{5} \\ 2 \end{bmatrix}\right) = \begin{bmatrix} -0.224 \\ 0.947 \end{bmatrix}$$

Now present I_3 :

$$a = f\left(W.I_3\right) = f\left(\begin{bmatrix} 0 & -1 \\ -0.947 & 0.224 \\ -0.224 & 0.947 \end{bmatrix}\begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}\right)$$

$$= f\left(\begin{bmatrix} -0.707 \\ -0.512 \\ 0.512 \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Third neuron won again:

$$W_{new}^{(3)} = W_{old}^{(3)} + \alpha\left(I_3 - W_{old}^{(3)}\right)$$

$$= \begin{bmatrix} 0.224 \\ 0.947 \end{bmatrix} + 0.5\left(\begin{bmatrix} -1/\sqrt{5} \\ 2/\sqrt{5} \end{bmatrix} - \begin{bmatrix} -0.224 \\ 0.947 \end{bmatrix}\right)$$

$$= \begin{bmatrix} -0.2417 \\ 0.8272 \end{bmatrix}$$

The final structure is now $W = \begin{bmatrix} 0 & -1 \\ -0.947 & 0.224 \\ 0.2417 & 0.8272 \end{bmatrix}$

Now try the following exercises.

E3) Train a competitive network using the following input pattern:
$I_1 = \begin{bmatrix} 1 & -1 \end{bmatrix}^T, I_2 = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$ and $I_3 = \begin{bmatrix} -1 & 1 \end{bmatrix}^T$. Use the Kohonen learning law with $\alpha = 0.5$ and train for one pass through the input patterns (Present each input once in order given.) Assume an initial weight matrix as $\begin{bmatrix} \sqrt{2} & 0 \\ 0 & \sqrt{2} \end{bmatrix}$.

Applications of Self-organizing Maps

1. The SOM is useful for vector quantization, clustering, feature extraction, and data visualization.
2. The SOM is well known for its ability to perform clustering while preserving topology.
3. The SOM was originally intended to approximate input signals or their PDFs, by quantified codebook vector that are localized in the input space to minimize a quantization error functional.
4. The SOM is able to divide the input space into region with common nearest reference vectors.
5. The SOM is related to adaptive C-means, but performs a topological feature map that is more complex than just cluster analysis.
6. The SOM is especially powerful for the visualization of high-dimensional data.
7. The SOM can be used to decompose complex information-processing systems into a set of simple subsystems.

Now, let us summarize the unit.

10.7 SUMMARY

In this unit, we have covered the following points.

1. The concepts and principles of competitive learning, which is the basis of Kohonen model of artificial neural networks were discussed. A list of the goals of the competitive learning gives a broader understanding of error minimization, entropy maximization, features maps, clustering, density estimation, etc.
2. The major part of this unit presents various aspects of Kohonen Network, namely the structure, normalization of the input, weight initialisation and weight updates and finally the error calculation.
3. Some attention is also devoted to Self-Organization feature Maps (SOM) in order to give you idea of the processes of Competition, Cooperation, and Synaptic Adaptation, which operate in SOM. Finally one graphical example of Kohonen network map is presented for a clearer understanding of the this class of network.

10.8 SOLUTIONS/ANSWERS

E1) Any two goals described in Sec. 10.2 may be given.

E2) Kohenen network is trained in unsupervised mode and is presented with data but the correct output that corresponds to that data is not specified. This differentiates the kohonen network from the feed forward backward propagation neural networks.

10.9 PRACTICAL ASSIGNMENT

Session 9

Write a program in ‘C’ language to find the updated/ modified weights for Kohonen Networks. Also test your program on the input patterns given in Example 6.

REFERENCE

1. Neural Networks in a Soft Computing Framework, K.-L.DU, M.N.S. Swamy.
2. Soft Computing, D.K. Pratihar.
3. Neuro-Fuzzy and Soft Computing, J.S.R. Jang, C.-T. Sun, E. Mizutani.
4. Neural Network, M. Ananda Rao, J. Srinvas.
5. Neural Network! A Classroom Approach, Satish, Tata McGraw-Hill.
6. http://highered.mcgraw_hill.com/site/0070482926.
7. Discovering knowledge in data – an introduction in data mining.
8. www.rgu.ac.ux/files/chapter7-hopfield.pdf
9. www.comp.leeds.ac.yx/az23/reading/Hopfield.pdf.