Students' Journal
Vol. 35, No. 1&2 January-June 1994, pp 105-125

Neural Computing: An Introduction and Some Applications

ASHISH GHOSH, NIKHIL R PAL AND SANKAR K PAL, FIETE

Machine Intelligence Unit, Indian Statistical Institute, 203 B T Road, Calcutta 700 035, India.

Artificial neural network models have been studied for many years with the hope of designing information proeessing systems to produee human-like performance. The present artiele provides an introduction to neural computing by reviewing three commonly used models (namely, Hopfield's model, Kobonen's model and the Multi-layer perceptron) and showing their applications in various fields such as pattern classification, image processing and optimization problems. Some discussions are made on the merits of fusion of fuzzy logic and neural network technologies, and robnstness of network based systems.

THE primary task of all neural systems is to control various biological functions. A major part of it is connected with behaviors, *ie*, controlling the state of the organism with respect to its environment. Human being can do it almost instantaneously and without much effort because of the remarkable processing capabilities of the nervous system. For example, if we look at a scene or listen to a music, we can easily (without conscious effort and time) recognize it. This high performance rate must be derived, at least in part, from the large number of neurons (roughly 10^{11}) participating in the task and also from the huge connectivity (thereby providing feedback) among them. The neurons participating operate in parallel and try to pursue a large number of hypotheses simultaneously thereby providing output almost in no time.

Artificial Neural Network (ANN) models [1-14] try to simulate the biological neural network/ nervous system with electronic circuitry. ANN models have been studied for many years with the hope of achieving human like performance (artificially), particularly in the field of speech and image recognition. The models are extreme simplifications of the actual human nervous system. The computational elements (called neurons/nodes/processors) are analogous to the fundamental constituents (called the neurons) of the biological nervous system. The biological neurons have a large number of connectivity to its neighbors via synapses, whereas in the electronic neural model the connectivity is very low. Like biological neurons the artificial neurons also get input from different sources, combine them and provide a single output. The topological organization of these neurons tries to emulate the organization of the human nervous system.

Figure 1 shows the structure of a biological neuron. Such a neuron gets input via synaptic connections from a large number of its neighbors. The accumulated input is then transformed to provide a single output which is transmitted to other neurons through the axon. If the total input is greater than some threshold θ, then the neuron transmits output to others and is said to have fired. Total amount of output given by the neuron is measured by its firing rate, *ie*, the number of times the neuron fires per unit time.

A schematic representation of an electronic

Paper No. 35-D;

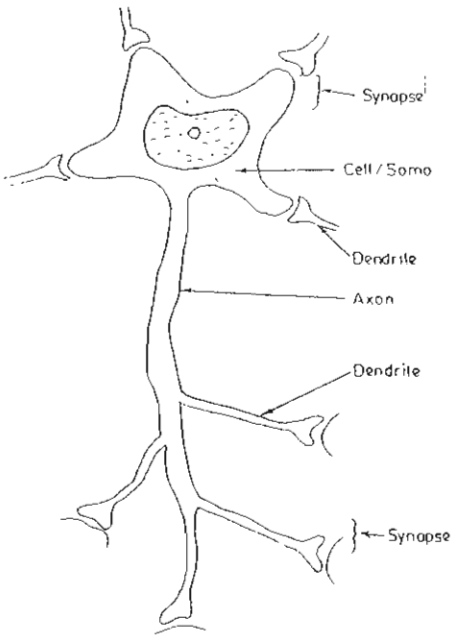


Fig 1 Biological neuron

neuron is given in Fig 2. The neuron gets input through the resistors from other neighboring neurons. By the operational amplifier the total input is then converted to a single output (summing the input and then amplifying it) and passed on to other neurons. So, as far as the primitive type of operations are concerned, an electronic neuron emulates a biological neuron. The artificial neurons are then connected to form a network structure which topologically as well as functionally tries to emulate the human nervous system. The resistance value by which a neuron is connected to another determines the connection strength (weight) between the neurons. The larger is the resistance, the smaller is the connection strength and vice versa.

LEARNING PARADIGM

The learning situation in neural networks can be broadly categorized into following two distinct types.

(*i*) Associative learning,
(*ii*) Regularity detector.

Associative Learning: In associative learning the system learns to produce a particular pattern of activation on one set of units whenever another pattern appears on another set of units. This further can be classified into

- autoassociator, and
- hetero/pattern associator.

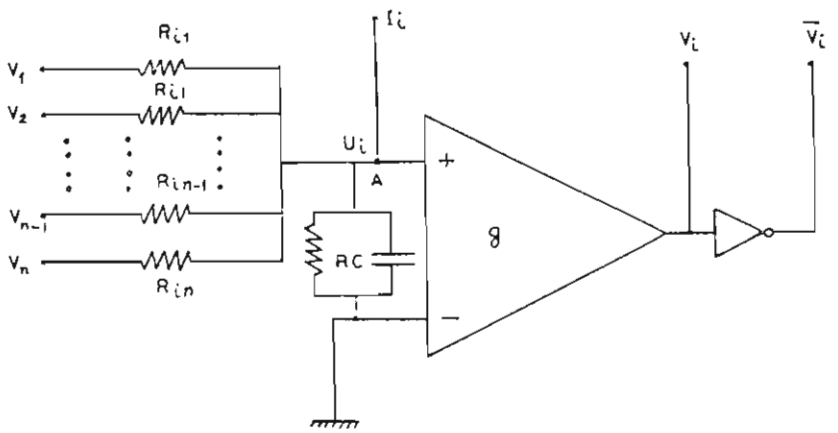


Fig 2 Electronic neuron

In auto associator a set of patterns is repeatedly presented and the system is supposed to store the patterns. Then, later, a part of one of the original patterns or possibly a pattern similar to one of the original patterns is presented and the task is to 'retrieve' the original pattern through a kind of pattern completion. This is an autoassociative process in which a pattern is associated with itself so that a degenerated version of the original pattern can act as a retrieval cue. Essentially it acts as a content addressable memory (CAM).

In hetero/pattern associator a set of pairs of patterns is repeatedly presented for training or learning of the network parameter. The system is to learn that when one member of the pair is presented, it produces the other. In this paradigm one seeks a mechanism in which an essentially arbitrary set of input patterns can be paired with an arbitrary set of output patterns. Therefore, associative learning is supervised where a set of training samples (samples of known classes/origin) is first of all used to learn the network parameters and then based on this learning it provides output corresponding to some new input.

Regularity detector

In this type of learning the system learns to respond to 'interesting' patterns in the input. In general, such a scheme should be able to form the basis for the development of feature detectors and should discover statistically salient features of the input population. Unlike the previous learning methods, there is no prior set of categories into which the patterns are to be classified. Here, the system must develop its own featural representation of the input stimuli which captures the most salient features of the population of the input pattern. In this sense, this learning is unsupervised because this does not use label samples for learning the network parameters. It may be mentioned that this type of learning resembles the biological learning to a great extent.

GENERAL DESCRIPTION OF NEURAL NETWORKS

ANN models [1,3,4,15,16] attempt to capture the key ingredients responsible for the remarkable capabilities of the human nervous system. A wide variety of models has been proposed for a variety of purposes; they differ in structure and details, but share some common features.

The common features are :

- Each node/neuron/processor is usually described by a single real variable (called its state), representing its output.
- The nodes are densely connected so that the state of one node affects the potential (total input) of many other nodes according to the weight or strength of the connection.
- The new state of a node is a non-linear function of the potential created by the firing activity of other neurons.
- Input to the network is given by setting the states of a subset of the nodes to specific values initially.
- The processing takes place through the evolution of the states of all the nodes of the net according to the dynamics of a specific net and continues until it stabilizes.
- The training (learning) of the net is a process whereby the values of the connection strengths are modified in order to achieve the desired output.

An artificial neural network can formally be defined as: Massively parallel interconnected network of simple (usually adaptive) processing elements which are intended to interact with the objects of the real world in the same way as biological systems do.

Neural networks are designated by the network topology, connection strength between pairs of neurons (weights), node characteristics and the status updating rules. Node characteristics mainly specify the primitive types of operations it can perform, like summing the weighted inputs coming to it and then amplifying it. The updating rules may be for that of weights and/or states of the processing elements (neurons). Normally an objective function is defined which represents the complete status of the network and the set of minima of it gives the stable states of the network. The neurons operate asynchronously (status of any neuron can be updated at random times independent of the other) or synchronously (parallelly) thereby providing output in real time. Since there are interactions among the neurons the collective computational property inherently reduces the computational task and makes the system robust. Thus ANNs are most suitable for tasks where collective decision making is required.

ANN models are reputed to enjoy the following four major advantages

* adaptivity-adjusting the connection strengths to new data/information
* speed–due to massively parallel architecture;
* robustness–to missing, confusing, ill-defined/ noisy data;
* optimality–as regards error rates in classification.

Though there are some common features of the different ANN models, they differ in finer details and in the underlying philosophy. Among the different models, mention may be made of the following three.

(*i*) Hopfield's model of associative memory. (autoassociator/CAM),

(*ii*) Kohonen's model of self-organizing neural network. (regularity detector/unsupervised classifer),

(*iii*) Multi-layer perceptron (hetero associator/supervised classifer).

In the following sections we will be describing three popularly used network models.

Hopfield's model

J J Hopfield [7-9] proposed a neural network model which functions as an auto-associative memory. So, before going into the details of the dynamics of the network and how it acts as an associative memory, we should first specify how we can construct an associative memory (see e.g. refer [3,4]).

Let there be given an associated pattern pair (x_k, y_k) where x_k is an N_1 dimensional vector and y_k is an N_2 dimensional column vector. The matrix,

$$M = y_k\, x_k^t \tag{1}$$

can serve as an associative memory where $y_k\, x_k^t$ is the direct product/dyadic product between the two vectors. The retrieval of the pattern y_k can be obtained when the input x_k is given, using,

$$\begin{aligned} Mx_k &= y_k\, x_k^t\, x_k \\ &= (x_k^t . x_k)\, y_k \\ &= y_k \qquad [\text{if } x_k^t . x_k = 1]. \end{aligned} \tag{2}$$

For illustration, let us consider the vector x_k and y_k to be of 3-dimensional.

Then,

$$M = y_k\, x_k^t = \begin{pmatrix} y_{k1}\, x_{k1} & y_{k1}\, x_{k2} & y_{k1}\, x_{k3} \\ y_{k2}\, x_{k1} & y_{k2}\, x_{k2} & y_{k2}\, x_{k3} \\ y_{k3}\, x_{k1} & y_{k3}\, x_{k2} & y_{k3}\, x_{k3} \end{pmatrix}$$

Retrieval is as follows,

$$Mx_k = \begin{pmatrix} y_{k1}x_{k1} & y_{k1}x_{k2} & y_{k1}x_{k3} \\ y_{k2}x_{k1} & y_{k2}x_{k2} & y_{k2}x_{k3} \\ y_{k3}x_{k1} & y_{k3}x_{k2} & y_{k3}x_{k3} \end{pmatrix} \begin{pmatrix} x_{k1} \\ x_{k2} \\ x_{k3} \end{pmatrix}$$

$$= (x^2_{k1} + x^2_{k2} + x^2_{k3}) \begin{pmatrix} y_{k1} \\ y_{k2} \\ y_{k3} \end{pmatrix}$$

$$= \begin{pmatrix} y_{k1} \\ y_{k2} \\ y_{k3} \end{pmatrix} \quad \text{if } (x^2_{k1} + x^2_{k2} + x^2_{k3}) = 1.$$

If the stimulus is not x_k but is x'_k, a distorted form of x_k, then the response is,

$$Mx'_k = y_k\, x^t_k\, x'_k \qquad (3)$$
$$= (x^t_k \cdot x'_k)\, y_k.$$

So, y_k will be recovered in undistorted form with amplitude $(x_k \cdot x'_k)$. In case of a set of associated pairs, the associative memory is formed as,

$$M = \sum_k y_k\, x^t_k. \qquad (4)$$

Recall of y_m on presentation of stimulus x_m is obtained as,

$$Mx_m = \left(\sum_k y_k\, x^t_k\right) x_m$$

$$= (x^t_m \cdot x_m)\, y_m + \sum_{k \neq m} (x^t_k \cdot x_m)\, y_k \qquad (5)$$

$$= y_m \quad \text{if } (x^t_k \cdot x_m) = \delta_{km},$$

the Kronecker delta. From eqn (5), a perfect recall is possible if the stored patterns x_k constitute a set of orthogonal basis vectors spanning the N_1 dimensional space. For such pattern $(x^t_k \cdot x_m) = 0$ when $k \neq m$ and $(x^t_m \cdot x_m) = 1$.

If the actual stimulus is a distorted version of the original stored pattern, that is if we have x'_m instead of x_m then the regenerated response will be a degraded version of y_m.

Hopfield's model of ANN acts as an associative memory storing the patterns in the above mentioned fashion. The network model consists of completely connected networks of idealized neurons. Here the cross products of the input patterns (vectors) are represented as the weights linking two nodes (whereas in matrix associative memory it becomes the elements of the matrix). The details of the model will be given later. In the associative memory we can store and retrieve several patters. In case of a network or physical system, the stored patterns can be represented as locally stable states of the system. So any physical system whose dynamics in the state space is dominated by a substantial number of locally stable states can be regarded as an associative memory/content addressable memory (CAM). Consider a physical system described by a vector **X**. Let the system have locally stable points $\boldsymbol{X}_a$, $\boldsymbol{X}_b$,....., $\boldsymbol{X}_n$. Then, if the system is started sufficiently near to any point $\boldsymbol{X}_a$, say at $\boldsymbol{X}_a + \Delta$, it will proceed in time until $X = X_a$. We can regard the information stored in the system as the vectors $\boldsymbol{X}_a$, $\boldsymbol{X}_b$, ..., $\boldsymbol{X}_n$. The starting point $\boldsymbol{X} = \boldsymbol{X}_a + \Delta$, represents a partial knowledge of the item $\boldsymbol{X}_a$, and the system then generates the total information $\boldsymbol{X}_a$. The dynamics of such a network governing the states and the status updating rules can either be discrete or continuous as described below.

Case 1: Discrete model

In this model two state neurons are used. Each neuron i, has two possible states, characterized by the output V_i of the neuron having values -1 or $+1$.

The input to a neuron comes from two sources, external input bias I_i and inputs from other neurons (to which it is connected). The total input to a neuron i is,

$$U_i = \sum_j W_{ij} V_j + I_i \qquad (6)$$

where W_{ij} (the weight) is the synaptic interconnection strength from neuron i to neuron j.

Useful computations in this system involve change of state of the system with time. So the motion of the state of the neural network in state space describes the computation that it performs. The present model describes it in terms of a stochastic evolution. Each neuron samples its input at any random time. It changes the value of its output or leaves it fixed according to a threshold rule with the threshold θ_i. The rule is,

$$V_i \rightarrow -1 \text{ if } U_i \leq \theta_i$$
$$\rightarrow 1 \text{ if } U_i > \theta_i . \qquad (7)$$

The interrogation of each neuron is a stochastic process, taking place at a fixed mean rate for each neuron. The times of interrogation of each neuron are independent of that of others thereby making the algorithm asynchronous.

The objective function (called the energy of the network) representing the overall status of the network is taken as,

$$E = -\sum_i \sum_j W_{ij} V_i V_j - \sum_i I_i V_i + \sum_i \theta_i V_i \qquad (8)$$

For a particular set up (fixed I_i, W_{ij}, θ_i), the energy E is a function of the status V of the nodes. The local minima of this function correspond to the stable states of the network. In this respect, the energy function E is a Liapunov function. A function $E(V)$ is said to be a Liapunov function in V^* if it satisfies the following properties

$\exists$ a δ such that letting $O(V^*,\delta)$ be the open neighborhood of V^* of radius δ, we have,

i) $E(V) > E(V^*)\ \forall V \in O(V^*,\delta),\ V \neq V^*$,

ii) $\dfrac{dE}{dt}(V^*) = 0$, and

iii) $\dfrac{dE}{dt}(V) < 0\ \forall V \in O(V^*, \delta),\ V \neq V^*$.

If the state V_i of the neuron i is changed to $V_i + \Delta V_i$ by the thresholding function (7), then the change ΔE in the energy E (8) is,

$$\Delta E = -\left[\sum_j W_{ij} V_j + I_i - \theta_i\right] \Delta V_i . \qquad (9)$$

Now if ΔV_i is positive then from the thresholding rule we notice that the bracketed quantity in eqn (9) is also positive, making ΔE negative. Similarly, when ΔV_i is negative, the bracketed quantity becomes negative resulting again ΔE negative. Thus any change in E under the previous thresholding rule (7) is negative. Since E is bounded, the time evolution of the system is a motion in the state space that seeks out minima in E and stops at such points. So given any initial state, the system will automatically update its status and will reach a stable state. The condition which guarantees convergence flow is,

$$W_{ij} = W_{ji} \text{ and } W_{ii} = 0;$$

ie, the weight matrix is symmetric and zero diagonal.

The previously mentioned model can be used as a content addressable memory (CAM) or associative memory. The operation of the net is as follows. First give the exemplars for all pattern classes. During this training the weights between different pairs of neurons get adjusted. Then an unknown pattern is

imposed on the net at time zero. Following this initialization, the net iterates in discrete time steps using the given formula. The net is considered to have converged when the outputs no longer change on successive iterations. The pattern specified by the node outputs after the convergence is the actual output. The above procedure can be written in algorithmic form as follows.

Step 1: Assign connection weights as,

$$W_{ij} = \begin{cases} \sum_{s=1}^{M} x_i^s x_j^s & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases}$$

W_{ij} is the connection strength between node i and j. x_i^s is the ith component of the sth pattern (assuming there are M N-dimensional patterns).

Step 2 : Present the next unknown pattern,

$$V_i(0) = x_i, \quad 1 \leq i \leq N$$

Here $V_i(t)$ is the output of node i at time index t.

Step 3: Iterate until convergence,

$$V_i(t+1) = g\left(\sum_{j=1, j\neq i}^{N} W_{ij} V_j(t)\right).$$

The function $g(.)$ is the thresholding function (eqn (7)). The process is iterated until the network stabilizes. At the converged stage the node outputs give the exemplar pattern that best matches the unknown pattern.

Step 4 : Repeat by going to Step 2.

Case 2 : *Continuous model*

This model is based on continuous variables and responses but retains all the significant behaviors of the discrete model. Let the output variable V_i for ith neuron lie in the range $[-1, +1]$, and be continuous and monotonically increasing (non linear) function of the instantaneous input U_i to it. The typical input/output relation

$$V_i = g(U_i) \tag{10}$$

is usually sigmoidal with asymptotes -1 and $+1$. It is also necessary that the inverse of g exists *ie*, $g^{-1}(V_i)$ is defined.

A typical choice of the function g is (Fig 3),

$$g(x) = 2\,\frac{1}{1 + e^{-(x-\theta)/\theta_0}} - 1. \tag{11}$$

Here the parameter θ controls the shifting of the function g along the x axis an θ_0 determines the steepness (sharpness) of the function. Positive values of θ shift $g(x)$ towards positive x axis and vice versa. Lower values of θ_0 make the function steeper while higher values make it less sharp. The value of $g(x)$ lies in $(-1, 1)$ with 0.0 at $x = \theta$.

Let for the realization of the gain function g we use a non linear operational amplifier with

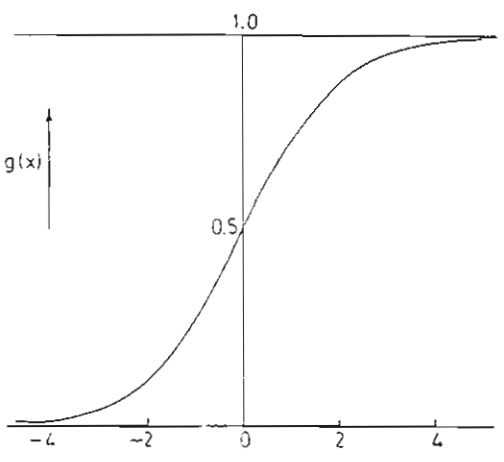


Fig3 Sigmoidal transfer function with threshold $\theta = 0$ and sharpness $\theta_0 = 1$

negligible response time. A possible electrical circuit for the realization of a neuron is shown in Fig 2. In the diagram V_1, V_2 ,...,V_n represent the output voltage (status) of the amplifiers (neurons) to which the ith amplifier is connected. R_{ij} $(=1/W_{ij})$ is the connecting resistance between the ith and the jth amplifiers. I_i is the input bias current for the ith amplifier. R is the input resistance of the amplifier and C the input capacitance of the same.

Suppose U_i is the total input voltage (potential at the point A, Fig 2) to the amplifier having a gain function g. Then applying Kirchoff's current law at the point A we get,

$$\sum_{j=1}^{n} \frac{V_j - U_i}{R_{ij}} + I_i = \frac{U_i}{R} + C \frac{dU_i}{dt}$$

$$\text{or, } C \frac{dU_i}{dt} = \sum_{j=1}^{n} \frac{V_j}{R_{ij}} - \sum_{j=1}^{n} \frac{U_i}{R_{ij}} + I_i - \frac{U_i}{R}$$

$$= \sum_{j=1}^{n} \frac{V_j}{R_{ij}} - \frac{U_i}{R_i} + I_i$$

$$\left[\frac{1}{R_i} = \sum_{j=1}^{n} \frac{1}{R_{ij}} + \frac{1}{R}\right]$$

$$= \sum_{j=1}^{n} W_{ij} V_j - \frac{U_i}{R_i} + I_i. \quad (12)$$

Here R_i is the total input resistance of the amplifier. The output impedance of the amplifier is considered to be negligible.

Now let us consider the quantity,

$$E = -\sum_i \sum_j W_{ij} V_i V_j - \sum_i V_i I_i + \sum_i \frac{1}{R_i} \int_0^{V_i} g^{-1}(V)\, dV. \quad (13)$$

Now,

$$\frac{dE}{dt} = \sum_i \frac{\partial E}{\partial V_i} \frac{dV_i}{dt}$$

$$= \sum_i \left(-\sum_j W_{ij} V_j - I_i + \frac{U_i}{R_i}\right) \frac{dV_i}{dt}$$

$$= \sum_i \left(-C \frac{dU_i}{dt}\right) \frac{dV_i}{dt} \quad \text{(from eqn (12))}$$

$$= -\sum_i \{C(g^{-1})'\left(\frac{dV_i}{dt}\right)^2\}.$$

Since g is a monotonic increasing function and C is a positive constant, every term in $\{.\}$ of the above expression is non negative, and hence, $\frac{dE}{dt} \leq 0$, and $\frac{dE}{dt} = 0 \rightarrow \frac{dV_i}{dt} = 0 \ \forall i$.

The function E is thus seen to be a Liapunov function. The function is also bounded. As $dE/dt \leq 0$, the searching in the direction of gradient will lead us to a minimum of E. So the evolution of the system is a motion in the state space that seeks out minima in E and stops at such points. Thus here also we can infer that given any initial state, the system will automatically update its status and will reach a stable state. The last term in eqn (13) is the energy loss term of the system. For a high gain of g (the input output transfer function), this term vanishes.

The network model with continuous dynamics is mainly used for optimization of functions. Only thing one has to do is to establish a correspondence between the energy function of a suitable network and the function to be optimized. In such optimization task it is likely that one will be trapped in some

local optimum. To come out of that, simulated annealing [17] can be used *ie*, run the network once again by giving perturbation from that local stable state. Repeatedly doing this one can obtain the absolute optimum. Hopfield and Tank [9] showed that ANNs can be used to solve computationally hard problems, in particular they applied it to a typical example of the Travelling Salesman Problem. Two applications of Hopfield's ANN model will be shown later.

The previously mentioned network has two major drawbacks when used as a content addressable memory. First, the number of patterns that can be stored and accurately retrieved is severely limited. If too many patterns are stored, the network many converge to a spurious pattern different from the exemplar pattern. The number of patterns that can be stored and retrieved perfectly is called the capacity of the network. The asymptotic capacity of such a network [18] is $n/4\log n$, where n is the number of neurons. A second limitation of this model is that an exemplar pattern will be unstable (*ie*, if it is applied at time zero it will converge to some other exemplar) if it shares many bits in common with another exemplar pattern, in other words the input patterns are not orthogonal. This shows that the orthogonality assumption of the storing patterns is a must (as in the matrix associative memory) for perfect retrieval of the stored pattern.

An extension of this associative memory is made by Kosko [19]. He introduced the concept of bi-directionality, forward and backward associative search. It was shown that any real matrix representing the connection strength or weights of a network can be considered as a bi-directional associative memory. This overcomes the symmetricity (of the weight matrix) assumption of Hopfield's model. Kosko's model is basically a two layered non-linear feed back dynamical system. In contrast to Hopfield's model (which acts as auto-associator) the present model acts as hetero associative CAM storing and recalling different pattern vectors.

Kohonen's model of self-organized feature mapping

Self-organization refers to the ability to find out the structure in the input data even when there is no prior information available about the structure. For example, if we look at a picture, we can very easily distinguish which portion of it belongs to object and which portion to background. Though we did not have prior knowledge of the structure of the object, we can find it out because of the self organizing ability of our nervous system. Kohonen's self-organizing network consists of a single layer of neurons as shown in Fig 4. The neurons, however, are highly interconnected within the layer as well as with the outside world. Let the signal

$$U = \{u_1, u_2, \ldots, u_n\}' \in R^n \qquad (14)$$

be connected as input to all the processing units. Suppose the length of the input vector be fixed (to some constant length) and the unit (processor/neuron) i have the weights (chosen as random numbers),

$$W_i = (\omega_{i1}, \omega_{i2}, \ldots, \omega_{in}) \qquad (15)$$

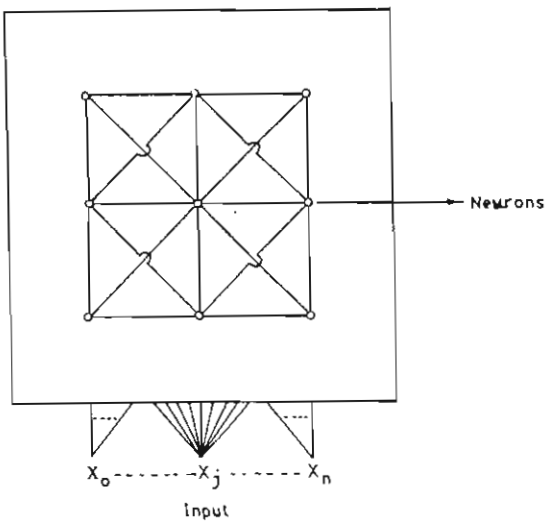


Fig 4 Kohonen's model of neural network

where, ω_{ij} is the weight of the ith unit from jth component of the input. The ith unit then forms the discriminant function,

$$\eta_i = \sum_{j=1}^{n} \omega_{ij} u_i = W_i . U \qquad (16)$$

where, η_i is the output of that unit. A discrimination mechanism will then detect the maximum of η_i. Let,

$$\eta_k = \max_i \{\eta_i\}. \qquad (17)$$

For the kth unit and all of its neighbors (within a specified radius of r, say) the following adaptation rule

$$W_k (t+1) = \frac{W_k (t) + \alpha . U(t)}{\| W_k(t) + \alpha . U(t) \|} \qquad (18)$$

is applied. Here the variables have been labeled by a discrete time index t, $\alpha (0 < \alpha < 1)$ is a gain parameter that decreases with time and the denominator is the length of the weight vector (which is used only for normalization of the weight vector). Note that the process does not change the length of W_k, rather rotates W_k towards U. Note that the decrement of α with time guarantees the convergence of the adaptation of the weights when sufficient input vectors are presented. This type of learning is known as competitive learning (regularity detector/unsupervised learning/clustering). Here all the processors try to match with the input signal, but the one which matches most (winner) takes the benefit of updating the weights (sharing with its neighbors). The basic idea is winner takes all. In this context it may be mentioned that sometimes the learning algorithm is modified in a way such that the nodes whieh had won the competition for a large number of times (greater than some fixed positive number) are not allowed to take part in the competition.

The concept can be written in algorithmic form as follows

Step 1: Initialize weights of the neurons to random values.

Step 2: Present new input

Step 3: Compute match η_i between the input and each output node i using

$$\eta_i = \sum_j W_{ij} . U_j .$$

Step 4: Select output node i^* as the output node with maximum η_i.

Step 5: Update weights to node i^* and neighbors using eqn (18).

Step 6: Repeat by going to step 2.

When sufficient number of inputs are presented, the weights will be stabilized and the clusters will be formed.

The basic points to be noted here are as follows:

- Continuous valued inputs are presented.
- After enough input vectors have been presented, weights will specify clusters.
- The weights will organize in such a fashion that the topologically close nodes/neurons/processors will be sensitive to inputs that are physically close.
- The weight vectors are normalized to have constant length.
- A set of neurons (may differ from cluster to cluster and from network to network) is allocated for a cluster.

The algorithm for feature map formation requires a neighborhood to be defined around the neurons. The size of the neighborhood gradually decreases with time. An application of Kohonen's

model to image processing (unsupervised classification/image segmentation) is described later. Application to supervised classification can also be found in [20] for speech recognition.

Multi-layer perceptron

A concept central to the practice of Pattern Recognition (PR) [21,22] is that of discrimination. The idea is that a pattern recognition system learns adaptively from experience and does various discrimination, each appropriate for its purpose. For example, if only belongingness to a class is of interest, the system learns from observations of patterns that are identified by class label and infers a discriminant for classification.

One of the most exciting developments during the early days of pattern recognition was the perceptron [23]. It may be defined as a network of elementary pocessors arranged in a manner reminiscent of biological neural nets which will be able to learn how to recognize and classify patterns in an autonomous manner. In such a system the processors are simple linear elements arranged in one layer. This classical (single layer) perceptron, given two classes of patterns, attempts to find a linear decision boundary separating the two classes. If the two sets of patterns are linearly separable, the perceptron algorithm is guaranteed to find a separating hyper plane in a finite number of steps. However, if the pattern space is not linearly separable, the perception fails and it is not known when to terminate the algorithm in order to get a reasonably good decision boundary. Keller and Hunt [24] attempted to provide a good stopping criterion (may not estimate a good decision boundary) for linearly non separable classes by incorporating the concept of fuzzy set theory into the learning algorithm of the perceptron. Thus, a single layer perceptron is inadequate for situations with multiple classes and non linear separating boundaries. This motivated the invent of the multi-layer network with non linear learning algorithms known as the multi-layer perceptron (MLP) [1].

A schematic representation of a multi-layer perceptron (MLP) is given in Fig 5. In general, such a network is made up of sets of nodes arranged in layers. Nodes of two different consecutive layers are connected by links or weights, but there is no connection among the elements of the same layer. The layer where the inputs are presented is known as input layer. Similarly, the layer producing output is called output layer. The layers in between the input and the output layers are known as hidden layers. The output of nodes in one layer are transmitted to nodes in another layer via links that amplify or attenuate or inhibit such outputs through weighting factors. Except for the input layer nodes, the total input to each node is the sum of weighted outputs of the nodes in the prior layer. Each node is activated in accordance with the input to the node and the activation function of the node. The total input to the ith unit of any layer is

$$U_i = \sum_j W_{ij} V_j . \qquad (19)$$

Here V_j is the output of the jth unit of the previous layer and W_{ij} is the connection weight between the ith node of one layer and the jth node of the previous layer. The output of a node

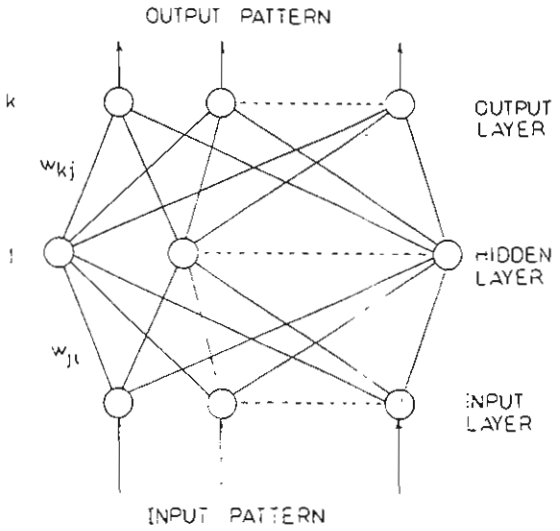


Fig 5 Multi-layer perceptron

i is $V_i = g(U_i)$, g is the activation function. Mostly the activation function is sigmoidal, with the form shown in equation (11) (Fig 3).

In the learning phase (training) of such a network we present the pattern $X = \{i_k\}$, where i_k is the kth component of the vector X, as input and ask the net to adjust its set of weights in the connecting links and also the thresholds in the nodes such that the desired output $\{t_k\}$ are obtained at the output nodes. After this we present another pair of X and $\{t_k\}$, and ask the net to learn that association also. In fact, we desire the net to find a simple set of weights and biases that will be able to discriminate among all the input/output pairs presented to it. This process can pose a very strenuous learning task and is not always readily accomplished. Here the desired output $\{t_k\}$ basically acts as a teacher which tries to minimize the error. In this context it should be mentioned that such a procedure will always be successful if the pattern is learnable by the MLP.

In general, the outputs $\{V_k\}$ will not be the same as the target or desired values $\{t_k\}$. For a pattern vector, the error is

$$E = \frac{1}{2} \sum_k (t_k - V_k)^2 \qquad (20)$$

where the factor of one half is inserted for mathematical convenience at some later stage of our discussion. The procedure for learning the correct set of weights is to vary the weights in a manner calculated to reduce the error E as rapidly as possible. This can be achieved by moving in the direction of negative gradient of E. In other words the incremental change ΔW_{kj} can be taken as proportional to $-\partial E / \partial W_{kj}$ for a particular pattern, *ie*, $\Delta W_{kj} = -\eta \, \partial E / \partial W_{kj}$.

However, E, the error is expressed in terms of the outputs $\{V_k\}$, each of which is a non-linear function of the total input to the node. That is,

$$V_k = g(U_k) \qquad (21)$$

where $U_k = \sum_j W_{kj} V_j$. Hence $\dfrac{\partial U_k}{\partial W_{kj}} = V_j$.

Now,

$$\Delta W_{kj} = -\eta \frac{\partial E}{\partial W_{kj}} = -\eta \frac{\partial E}{\partial U_k} \frac{\partial U_k}{\partial W_{kj}}$$

$$= -\eta \frac{\partial E}{\partial U_k} V_j = -\eta \frac{\partial E}{\partial V_k} \frac{\partial V_k}{\partial U_k} V_j$$

$$= -\eta \frac{\partial E}{\partial V_k} g'(U_k) V_j. \qquad (22)$$

Now if the kth element comes from the output layer then

$$\frac{\partial E}{\partial V_k} = -(t_k - V_k) \text{ (from eqn (20))}$$

and thus the change in weight is given by

$$\Delta W_{kj} = \eta (t_k - V_k) g'(U_k) V_j. \qquad (23)$$

If the weights do not affect the output nodes directly (for links between the input and the hidden layer, and also between two consecutive hidden layers), the factor $\partial E / \partial V_k$ can not be calculated so easily. In this case we use the chain rule to write

$$\frac{\partial E}{\partial V_k} = \sum_m \frac{\partial E}{\partial U_m} \frac{\partial U_m}{\partial V_k}$$

$$= \sum_m \frac{\partial E}{\partial U_m} \frac{\partial}{\partial V_k} \sum_i W_{mi} V_i = \sum_m \frac{\partial E}{\partial U_m} W_{mk}. \qquad (24)$$

Hence

$$\Delta W_{kj} = \begin{cases} -\eta \left(\dfrac{\partial E}{\partial V_k} \right) g'(U_k) V_j \\ -\eta \left(\sum_m \dfrac{\partial E}{\partial U_m} W_{mk} \right) g'(U_k) V_j \end{cases} \qquad (25)$$

for the output layer and other layers, respectively.

It may be mentioned here that a large value of η corresponds to rapid learning but might result in oscillations. A momentum term of $\alpha \Delta W_{kj}(t)$ can be added to avoid this and thus expression (22) can be modified as

$$\Delta W_{kj}(t+1)) = -\eta \frac{\partial E}{\partial V_k} g'(U_k) V_j + \alpha \Delta W_{kj}(t) \qquad (26)$$

where the quantity $(t + 1)$ is used to indicate the $(t+1)$th time instant, and α is a proportionality constant. The second term is used to ensure that the change in W_{kj} at $(t+1)$th instant should be somewhat similar to the change undertaken at instant t.

During training, each pattern of the training set is used in succession to clamp the input and output layers of the network. Feature values of the input patterns are clamped to the input nodes whereas the output nodes are clamped to class labels. The input is then passed on to the output layer via the hidden layers to produce output. Error is then calculated using eqn (2) and is back propagated to modify the weights. A sequence of forward and backward passes is made for each input pattern. A set of training samples is used several times for the stabilization (minimum error) of the network. After the network has been settled down the weight specify the boundaries of the classes. The learning of this type of network is therefore supervised. Since the error is back propagated for learning the parameters (weight correction) the process is also called back propagation learning. Next phase is testing/ recognition. Here we give an unknown input and the neuron with maximum output value will indicate the class to which it belongs.

APPLICATIONS

Artificial neural networks are suitable mainly for the problems where a collective decision making is required. Major areas in which the ANN models have been applied in order to exploit its high computational power and to make robust decisions are supervised classification [25,26] complex optimization problems (like travelling salesman problem [9] and minimum cut problems [14]), clustering [27], text classification [28], speech recognition [26], image compression [29-31], image segmentation [32], image restoration [33,34], object extraction [35-38] and object recognition [39]. Among them only four are described here in short (chosen from three typical areas such as classification, optimization and image processing).

Travelling salesman problem

For applying Hopfield type neural network solve optimization problems, what one has to do is to design a function (to be optimized) in the form of the energy function of this type of network and to provide an updating rule so that the system automatically comes to an optimum of the function and stops there.

One of the classic examples of computationally hard problems is the Travelling Salesman Problem (TSP). The problem can be described as follows: A salesman is to tour N cities, visit one city once and only once and return to the starting city. No two cities can be visited at the same time. The distances between the cities are known. There are $N!/2N$ distinct possible tours with different lengths. The question is what should be the order of visiting the cities so that the distance traversed is minimum?

The problem can be represented in terms of network structure as follows. The cities correspond to the nodes of the network, the strengths of the connecting links correspond to the distances between the cities. For an N city problem, let us consider a $N \times N$ matrix of nodes. If a node is ON (1), the city is visited, and if it is OFF (−1), the city is not visited. In the matrix let the row number correspond to

the city number, and the column number correspond to the time (order) when it is visited.

The energy function of the network which has to be minimized for the TSP problem is

$$E = \frac{A}{2} \sum_X \sum_i \sum_{j \neq i} V_{Xi} V_{Xj} + \frac{B}{2} \sum_i \sum_X \sum_{Y \neq X} V_{Xi} V_{Yi} + \frac{C}{2} \left(\sum_X \sum_i V_{Xi} - N \right)^2 + \frac{D}{2} \sum_X \sum_{Y \neq X} \sum_i d_{XY} V_{Xi} (V_{Y\,i+1} + V_{Y\,i-1}) - \sum_X \sum_i I_{Xi} V_{Xi} + \frac{1}{\tau} \sum_X \sum_i \int_0^{V_{xi}} g^{-1}(z)\, dz. \quad (27)$$

Here A,B,C,D are large positive constants. The X,Y indices refer to cities and i,j to the order of visiting. If $V_{Xi} = 1$, it means that the city X is visited at the ith step. When the network is searching for a solution, the neurons are partially active, which means that a decision is not yet reached. The first three terms of the above energy function correspond to the syntax *ie*, no city can be visited more than once, no two cities can be visited at the same time, and a total of N cities will only be visited. When the syntax is satisfied, E reduces to just the fourth term of it, which is basically the length of the tour with d_{XY} as the distance between the cities X and Y. Any deviation from the constraints will increase the energy function. The fifth term is due to the external bias. The last term is the energy loss, and τ is the characteristic decay time of the neurons.

To find a solution we select at random an initial state for the network and let it evolve according to the updating rule. It will eventually reach a steady state (a minimum of E) and stop. When the network has settled down, in each row and each column only one neuron is active. The neuron which is active in the first column shows the city which is visited first, the neuron in the second column shows the city visited second and so on. The sum of the weights (d_{XY}) gives the length of the tour (which is an optimum one). The energy E may have many minima, deeper minima correspond to short tours and the deepest minimum to the shortest tour. To get the deepest minimum one can use the simulated annealing [17] type of techniques.

Noisy image segmentation

Application of associative memory model for image segmentation is described below in short (this will be referred to as Algorithm 1 for the rest part of the article). For more details see [35]. The algorithm requires understanding of a few definitions which are given first of all. For further references one can refer [35-38,40-46].

2 D Random Field: A family of functions $f_{(x,y)}(q_i)$ over the set of all outcomes $Q = \{q_1, q_2, \ldots, q_m\}$ of an experiment, with (x,y) as a point in the 2-D Euclidean plane is called a 2-D random field.

Let us focus our attention on a 2-D random field defined over a finite $N_1 \times N_2$ rectangular lattice of points (pixels) characterized by,

$$L = \{(i,j) : 1 \leq i \leq N_1,\ 1 \leq j \leq N_2\} \quad (28)$$

Neighborhood system: The dth order neighborhood (N_{ij}^d) of any element (i,j) on L is described as,

$$N_{ij}^d = \{(i, j) \in L\}$$

such that

- $(i,j) \notin N_{ij}^d$
- if $(k, l) \in N_{ij}^d$ then $(i,j) \in N_{kl}^d$

and the neighborhood system $N^d = \{N^d_{ij}\}$.

Different ordered neighborhood systems can be defined considering different sets of neighboring pixels of (i,j). $N^1 = \{N^1_{ij}\}$ can be obtained by taking the four nearest neighbor pixels. Similarly, $N^2 = \{N^2_{ij}\}$ consists of the eight pixels neighboring (i,j) and so on. Due to the finite size of the used lattice (the size of the image being fixed), the neighborhood of the pixels on the boundaries are necessarily smaller unless periodic lattice structure is assumed.

Cliques: A clique for the dth order neighborhood system N^d defined on the lattice L, denoted by c, is a subset of L such that,

i) c consists of a single pixel, or,

ii) for $(i,j) \neq (k,l)$, $(i,j) \in c$ and $(k,l) \in c$ implies that $(i,j) \in N^d_{kl}$.

Gibbs distribution: Let N^d be the dth order neighborhood system defined on a finite lattice L. A random field $X = \{X_{ij}\}$ defined on L has a Gibbs distribution or equivalently is a Gibbs Random Field (GRF) with respect to N^d if and only if its joint distribution is of the form,

$$p(X = x) = \frac{1}{Z} e^{-U(x)} \qquad (29)$$

with

$$U(x) = \sum_{c \in C} V_c(x), \qquad (30)$$

where $V_c(x)$ is the potential associated with the clique c, C is the collection of all possible cliques and,

$$Z = \sum_{x} e^{-U(x)}. \qquad (31)$$

Here Z, the partition function, is a normalizing constant, and $U(x)$ is called the energy of the realization. The clique potential $V_c(x)$ is assumed to depend on the gray values of the pixels in c and on the clique types. The joint distribution expression in eqn (29) tells that with decrease in $U(x)$ the probability $p(X = x)$ increases. Since the clique potential can account for the spatial interaction of pixels in an image several authors have used this GRF for modeling scenes.

Here it is assumed that the random field X consists of M-valued discrete random variables $\{X_{ij}\}$ taking values in $Q = \{q_1, q_2, \ldots, q_m\}$. To define the Gibbs distribution it suffices to define the neighborhood system N^d and the associated clique potentials $V_c(x)$. It has been further found that the 2nd order neighborhood system is sufficient for modeling the spatial dependencies of a scene consisting of several objects. Hence we shall concentrate on the cliques associated with $N^{=2}$.

Gibbsian model for image data

A digital image $y = \{y_{ij}\}$ can be described as an $N_1 \times N_2$ matrix of observations. It is assumed that the matrix y is a realization from a random field $Y = \{Y_{ij}\}$. The random field Y is defined in terms of the underlying scene random field $X = \{X_{ij}\}$.

Let us assume that the scene consists of several regions each with a distinct brightness level and the realized image is a corrupted version of the actual one, corrupted by white (additive independent and identically distributed *ie*, additive iid,) noise. So Y can be written as,

$$Y_{ij} = X_{ij} + W_{ij} \quad \forall (i,j) \in L \qquad (32)$$

where W_{ij} is iid noise. It is also assumed that $W \sim N(0, \sigma^2)$.

Segmentation algorithm

Here the problem is simply the determination of the scene realization x that has given rise to the

actual noisy image y. In other words, we need to partition the lattice into M region types such that $X_{ij} = q_m$ if x_{ij} belongs to the mth region type. The realization x cannot be obtained deterministically from y. So the problem is to estimate $\hat{x}$ of the scene X, based on the realization y. The statistical criterion of maximum a'posteriori probability (MAP) can be used to estimate the scene. The objective in this case is to have an estimation rule which yields $\hat{x}$ that maximizes the a'posteriori probability $p(X = x \mid Y = y)$. Taking logarithm and then simplifying, the function to be maximized takes the form,

$$-\sum_{c \in C} V_c(x) - \sum_{m=1}^{M} \sum_{(i,j) \in S_m} \frac{1}{2\sigma^2} (y_{ij} - q_m)^2 \quad (33)$$

with $S_m = \{(i,j) \in L;\ x_{ij} = q_m\}$.

In the present case we shall consider scenes with two regions only, object and background (*ie*, $M = 2$). Without loss of generality, we can assume that in the realized image gray values are in the range -1 to $+1$. It is already known that in an image if a pixel belongs to region type k, the probability of its neighboring pixels to lie in the same region type is very high. This suggests that if a pair of adjacent pixels is having similar values then the potential of the corresponding clique should increase the value of $p(X = x)$, *ie*, the potential of the clique associated with these two pixels should be negative. If the values V_i and V_j of two adjacent pixels are of same sign, then $V_c(x)$ should be negative, otherwise it should be positive.

One possible choice of the clique potential $V_c(x)$, of a clique c of the realization x, containing ith and jth pixels can be

$$V_c(x) = -W_{ij} V_i V_j \quad (34)$$

where W_{ij} is a constant for particular i and j, and $W_{ij} > 0$. This W_{ij} can be viewed as the connection strength between the ith and jth neurons and V_i as the output of the ith neuron.

y_{ij} is the realization of the modeled value x_{ij} of the (i,j)th pixel and q_m is the value of the (i,j)th pixel in the underlying scene, *ie*, $x_{ij} = q_m$. For a neural network one can easily interpret x_{ij} as the present staus of the (i,j)th neuron and y_{ij} as the initial bias of the same neuron. Thus we can write I_{ij} for y_{ij} and V_{ij} for q_m. Substituting these in the objective function (after removing the constant terms) to be maximized we get (changing the two dimensional indices to one dimensional index),

$$\sum_{i,j} W_{ij} V_i V_j - \frac{1}{2\sigma^2} (-2 \sum_i I_i V_i + \sum_i V_i^2). \quad (35)$$

So our task reduces to the minimization of

$$-\sum_{i,j} W_{ij} V_i V_j - \frac{1}{\sigma^2} \sum_i I_i V_i + \frac{1}{2\sigma^2} \sum_i V_i^2 \quad (36)$$

Let us now consider a neural network with architecture as shown in Fig 6 and having a neuron realization as in Fig 7. The expression

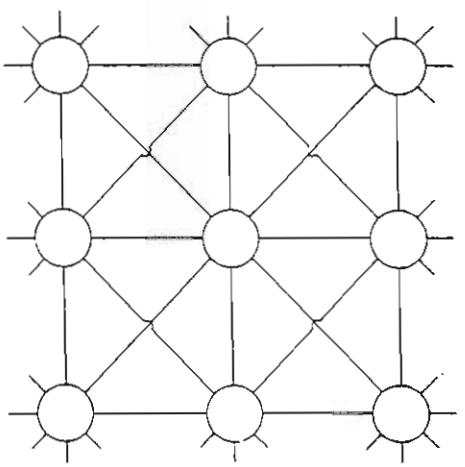


Fig 6 Neural network structure used for image segmentation

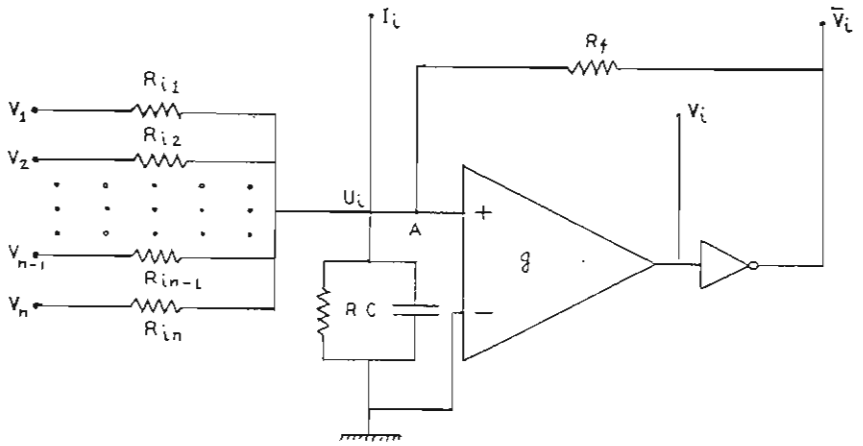


Fig 7 Electronic neuron used for image segmentation (algorithm 1)

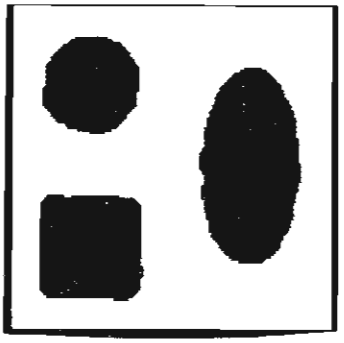
$$-\sum_{i,j} W_{ij} V_i V_j - \sum_i I_i V_i + \frac{1}{2R_f} \sum_i V_i^2 + \sum_i \frac{1}{R_i} \int_0^{V_i} g^{-1}(v)\, dv \qquad (37)$$

can be shown to be the energy function of this network. The last term in the above equation is the energy loss of the system which vanishes for very high gain of g. Equation (37) then reduces to,

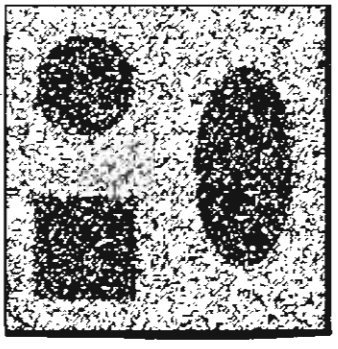
$$-\sum_{i,j} W_{ij} V_i V_j - \sum_i I_i V_i + \frac{1}{2R_f} \sum_i V_i^2 . \qquad (38)$$

The above expression is equivalent to our objective function (eqn (36)) with proper adjustment of coefficients. So the minima values of the objective function can be easily determined with the previously described neural network and hence the actual scene realization. It may be mentioned here that in this study W_{ij}'s are taken as unity (1).

As an illustration, in Fig 8*a* the object extracted from the noisy image shown in Fig 8*b* using Algorithm 1 is depicted.



(*a*)



(*b*)

Fig 8 (*a*) Input image (*b*) Extracted object by algorithm 1

Object extraction using self-organizing neural network

In this subsection we will be describing an application of Khonen's model of neural network in object extraction (image segmentation) problem which we will be referring as Algorithm 2. For greater details, see [37]. It may be mentioned here that in Algorithm 1 the image segmentation/object extraction problem is viewed as an optimization problem and is solved by Hopfield type neural network; but in the present algorithm the object extraction problem is treated as a self-organizing task and is solved by employing Kohonen type neural network architecture.

An L level $M \times N$ dimensional image $X(M \times N)$ can be represented by a neural network $Y(M \times N)$, having $M \times N$ neurons arranged in M rows and N columns (Fig 6). Let the neuron in the ith row and jth column be represented by $y(i,j)$ and let it take input from the global/local information of the (i,j)th pixel $x(i,j)$ having gray level intensity x.

It has already been mentioned that in feature mapping algorithm, the maximum length of the input and weight vectors is fixed. Let the maximum length for each component of the input vector be unity. To keep the value of each component of the input ≤ 1, let us apply a mapping,

$$f : [l_{min}, l_{max}] \rightarrow [0,1].$$

Here l_{max} and l_{min} are the highest and the lowest gray levels available in the image. The components of the weight vectors are chosen as random numbers in [0,1].

The input to a neuron $y(i,j)$ is considered to be an N-dimensional vector,

$$U(i,j) = [u_1(i,j),\ u_2(i,j),.....u_n(i,j)]'.$$

The weight (to a neuron) is also a vector quantity,

$$W(i,j) = [w_1(i,j),\ w_2(i,j),.....w_n(i,j)].$$

Then compute the dot product as,

$$d(i,j)_t = W(i,j)_t \,.\, U(i,j)_t = \sum_{k=1}^{n} w_k(i,j).\ u_k(i,j) \quad (39)$$

where the subscript t is used to denote the time instant.

Only those neurons for which $d(i,j)_t > T$ (T is assumed to be a threshold) are allowed to modify their weights along with their neighbors (within a specified radius). Consideration of a set of neighbors enables one to grow the region by including those elements which might have been dropped out because of the randomness of weights. The weight updating procedure is same as in equation (18). The value of α (eqn (18)) decreases with time. In the present case, it is chosen to be inversely related to the iteration number.

The output (just to check convergence) of the network is calculated as,

$$O_t = \sum_{i,j} d(i,j)_t \quad (40)$$

with

$$d(i,j)_t > T.$$

The procedure for updating of the weights is continued until,

$$|O_{t+1} - O_t| < \delta \quad (41)$$

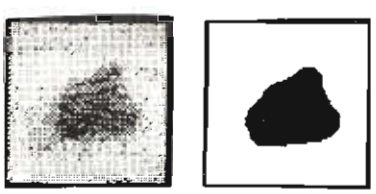
where δ is a pre-assigned very small positive quantity. At this stage the network is said to have converged (settled). After the network has converged, the pixels $x(i,j)$s (along with their neighbors) with

the constraint $d(i,j) > T$, may be considered to constitute a region, say, object.

In the above section, a threshold T in WU plane was assumed to decide whether to update (or not to do so) the weights of a neuron; *ie*, the threshold T divides the neuron set into two groups (thereby the image space into two regions, say object and background). The weights of one group is updated only. For a particular threshold the updation procedure continues until it converges. The threshold is then varied. The statistical criterion least square error or correlation maximization can be used for selecting the optimal threshold.

The extracted object from the image in Fig 9*a* by Algorithm 2 is given in Fig 9*b* for visual inspection.

Robustness with respect to component failure is one of the most important characteristics of ANN based information processing systems. Robustness analysis of the previously mentioned algorithms is available in [47]. This was performed by damaging some of the nodes/links and studying the performance of the system (*ie*, the quality of the extracted objects). The damaging process is viewed as a pure death process and is modeled as Poisson process. Sampling from distribution using random numbers has been used to choose the instants of damaging the nodes/links. The nodes/links are damaged randomly. The qualities of the extracted object regions were not seen to be much deteriorated even when $\simeq$ 10% of the nodes and links were damaged.



(*a*) (*b*)

Fig 9 (*a*) Input image (*b*) Extracted object by algorithm 2

Classification using multi-layer perceptron

Multi-layer perceptron finds its application mainly in the field of supervised classifier design. A few applications can be found in [25,26,48-50] applied to cloud classification, pattern classification, speech recognition, pixel classification and histogram thresholding of an image. Very recently an unsupervised classification using multi-layer neural network is reported by Ghosh *et al* [38].

An experiment for the classification of vowel sounds, using MLP has been described in [26]. The input patterns have the first three formant frequencies of the six different vowel classes and the outputs are the class numbers. The percentage of correct classification was $\simeq$ 84. In order to improve the performance (% of correct classification and capability of handling uncertainties in inputs/outputs) Pal and Mitra [26] have built a fuzzy version of the MLP by incorporating the concepts from fuzzy sets [42,51,52] at various stages of the network. Unlike the conventional system, their model is capable of handling inputs available in linguistic form (*eg*, small, medium and high). Learning is based on fuzzy class membership of the training patterns. The final classification output is also fuzzy.

At present many researchers have been tryi to fuse the concepts of fuzzy logic (which has shown enormous promise in handling uncertainties to a remarkable extent) and ANN technologies (proved to generate highly non-linear boundaries) in order to exploit their merits for designing more intelligent (human-like) systems. The fusion is mainly tried out in two different ways. The first way is to incorporate the concepts of fuzziness into an ANN framework and the second one is to use ANNs for a variety of computational tasks within the framework of pre-existing fuzzy models. Fuzzy sets have shown their remarkable capabilities in modeling human thinking process; whereas ANN models emerge with the promise of emulating the human nervous system-the controller of human thinking process. Fusion of

these two should materialize the creation of human like intelligent systems to a great extent. For further references one can refer [52-54].

REFERENCES

1. D E Rumelhart, J McClelland *et al, Parallel Distributed Processing : Explorations in the Microstructure of Cognition*, vol 1, Cambridge, MA: MIT Press, 1986.
2. D E Rumelhart, J McClelland *et al, Parallel Distributed Processing*, vol 2, Cambridge, MA: MIT Press, 1986.
3. Y H Pao, *Adaptive Pattern Recognition and Neural Networks*, New York: Addison-Wesley, 1989.
4. T Kohonen, *Self-organization and Associative Memory*. Berlin: Springer Verlag, 1989.
5. T Kohonen, An introduction to neural computing, *Neural Networks*, vol 1, pp 3-16, 1988.
6. R P Lippmann, An introduction to computing with neural nets, *IEEE ASSP Magazine*, pp 3-22, 1987.
7. J J Hopfield, Neural network and physical systems with emergent collective computational abilities, *Proceedings National Academy of Science* (USA), pp 2554-2558, 1982.
8. J J Hopfield, Neurons with graded response have collective computational properties like those of two state neurons, *Proceedings National Academy of Science* (USA), pp 3088-3092, 1984.
9. J J Hopfield & D W Tank, Neural computation of decision in optimization of problems, *Biological Cybernetics*, vol 52, pp 141-152, 1985.
10. P D Wassermann, *Neural Computing : Theory and Practice*, New York : Van Nostrand Reinhold, 1990.
11. L O Chua & L Yang, Cellular neural network : theory, *IEEE Transactions on Circuits and Systems*, vol 35, pp 1257-1272, 1988.
12. L O Chua & L Yang, Cellular neural networks : applications, *IEEE Transactions on Circuits and Systems*, vol 35, pp 1273-1290, 1988.
13. B M Forrest *et al*, Neural network models, *Parallel Computing*, vol 8, pp 71-83, 1988.
14. J Bruck & S Sanz, A study on neural networks, *International Journal of Intelligent Systems*, vol 3, pp 59-75, 1988.
15. G A Carpenter & S Grossberg, A massively parallel architecture for a self-organizing neural pattern recognition machine, *Computer Vision, Graphics and Image Processing*, vol 37, pp 54-115, 1987.
16. G A Carpenter, Neural network models for pattern recognition and assoeiative memeory, *Neural Networks*, vol 2, pp 243-257, 1989.
17. S Kirkpatric, C D Gellat & M P Vecchi, Optimization by simulated annealing, *Science*, vol 220, pp 671-680, 1983.
18. R J McEliece, E X Posner, E R Rodenmich & S S Venkatesh, The capacity of the Hopfield associative memory, *IEEE Transactions on Information Theory*, vol 33, pp 461-482, 1987.
19. B Kosko, Adaptive bidirectional associative memories, *Applied Optics*, vol 26, pp 4947-4960, 1987.
20. S Mitra & S K Pal, Self-organizing neural network as a fuzzy classifier, *IEEE Transactions on Systems, Man, and Cybernetics*, vol 24, 1994, (to appear).
21. J T Tou & R C Gonzalez, *Pattern Recognition Principles*, Reading, MA: Addison-Wesley, 1974.
22. R O Duda & P E Hart, *Pattern Classification and Scene Analysis*, New York: John Wiley, 1973.
23. M Minsky & S Papert, *Perceptrons*, Cambridge, MA:MIT Press, 1969.
24. J M Keller & D J Hunt, Incorporating fuzzy membership fnnctions into the perceptron algorithm, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol 7, pp 693-699, 1985.
25. R P Lippmann, Pattern classification using neural networks, *IEEE Communications Magazine*, pp 47-64, 1989.
26. S K Pal & S Mitra, Multilayer perceptron, fuzzy sets and classification, *IEEE Transactions on Neural Networks*, vol 3, pp 683-697, 1992.
27. B K Parsi & B K Parsi, On problem solving with Hopfield neural networks, *Biological Cybernetics*, vol 62, pp 415-423, 1990.
28. D J Burr, Experiments on neural net recognition of spoken and written text, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol 36, pp 1162-1168, 1988.
29. G W Cottrel, P Munro & D Zipser, Image compression by back propagation : an exemplar of extensional programming, *Tech Rep ICS*-8702, University of California, San Diego, 1987.

30. G W Cottrel & P Munro, Principal component analysis of images via back propagation, *SPIE : Visual Communication and Image Processing*, vol 1001, pp 1070-1077, 1988.

31. S P Luttrell, Image compression using a multilayer neural network, *Pattern Recognition Letters*, vol 10, pp 1-7, 1989.

32. G L Bilbro, M White & W Synder, Image segmentation with neuro-computers, *Neural Computers*, (R Eckmiller & C V D Malsberg, eds), Springer-Verlag, 1988.

33. L Bedini & A Tonazzini, Neural network use in maximum entropy image restoration, *Image and Vision Computing*, vol 8, pp 108-114, 1990.

34. Y T Zhou *et al*, Image restoration using a neural network, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol 36, pp 940-943, 1988.

35. A Ghosh, N R Pal & S K Pal, Image segmentation using a neural network, *Biological Cybernetics*, vol 66, pp 151-158, 1991.

36. A Ghosh & S K Pal, Neural network, self-organization and object extraction, *Pattern Recogniion Letters*, vol 13, pp 387-397, 1992.

37. A Ghosh, N R Pal & S K Pal, Object background classification using Hopfield type neural network, *International Journal of Pattern Recognition and Artificial Intelligence*, vol 6, pp 989-1008, 1992.

38. A Ghosh, N R Pal & S K Pal, Self-organization for object extraction using multilayer neural network and fuzziness measures, *IEEE Transactions on Fuzzy Systems*, vol 1, pp 54-68, 1993.

39. N M Nasrabadi & W Li, Object recognition by a Hopfield neural network, *IEEE Transactions on Systems, Man, and Cybernetics*, vol 21, 1991.

40. R C Gonzalez & P Wintz, *Digital Image Processing*, Reading, MA: Addison-Wesley, 1987.

41. A Rosenfeld & A C Kak, *Digital Picture Processing*, New York: Academic Press, 1982.

42. S K Pal & D D Majumder, *Fuzzy Mathematical Approach to Pattern Recognition*, New York : John Wiley (Halsted Press), 1986.

43. J Besag, Spatial interaction and statistical analysis of lattice systems, *Journal of Royal Statistical Society*, vol B-36, pp 192-236, 1974.

44. A C Cohen & D B Cooper, Simple parallel hierarchical and relaxation algorithms for segmenting noncausal Markovian field, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol 9, pp 195-219, 1987.

45. H Derin & H Elliott, Modeling and segmentation of noisy and textured images using Gibbs random fields, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol 9, pp 39-55, 1987.

46. S Geman & D Geman, Stochastic relaxation, Gibbs distribution, and the Bayesian restoration of images, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol 6, pp 721-741, 1984.

47. A Ghosh, N R Pal & S K Pal, On robustness of neural network based systems under component failure, *IEEE Transactions on Neural Networks*, (to appear).

48. J Lee, R C Weger, S K Sengupta & R M Welch, A neural network approach to cloud classification, *IEEE Transactions on Geoscience and Remote Sensing*, vol 28, pp 846-855, 1990.

49. W E Blanz & S L Gish, A real time image segmentation system using a connectionist classifier architeeture, *International Journal of Pattern Recognition and Artificial Intelligence*, vol 5, pp 603-617, 1991.

50. N Babaguchi, K Yamada, K Kise & Y Tezuku, Connectiouist model binarization, *International Journal of Pattern Recognition and Artificial Intelligence*, vol 5, pp 629-644, 1991.

51. L A Zadeh, Fuzzy sets, *Information and Control*, vol 8, pp 338-353, 1965.

52. J C Bezdek & S K Pal (eds), *Fuzzy Models far Pattern Recognition : Methods that Search for Structures in Data*, New York: IEEE Press, 1992.

53. Special issue on neural networks, *International Journal of Pattern Recognition and Artificial Intelligence*, vol 5, 1991.

54. Special issue on fuzzy logic and neural networks, *International Journal of Approximate Reasoning*, vol 6, 1992.