Sādhanā, Vol. 19, Part 2, April 1994, pp. 189–238. © Printed in India.

Artificial neural networks for pattern recognition

B YEGNANARAYANA

Department of Computer Science and Engineering, Indian Institute of Technology, Madras 600036, India

E-mail: yegna@iitm.ernet.in

MS received 12 April 1993; revised 8 September 1993

Abstract. This tutorial article deals with the basics of artificial neural networks (ANN) and their applications in pattern recognition. ANN can be viewed as computing models inspired by the structure and function of the biological neural network. These models are expected to deal with problem solving in a manner different from conventional computing. A distinction is made between pattern and data to emphasize the need for developing pattern processing systems to address pattern recognition tasks. After introducing the basic principles of ANN, some fundamental networks are examined in detail for their ability to solve simple pattern recognition tasks. These fundamental networks together with the principles of ANN will lead to the development of architectures for complex pattern recognition tasks. A few popular architectures are described to illustrate the need to develop an architecture specific to a given pattern recognition problem. Finally several issues that still need to be addressed to solve practical problems using ANN approach are discussed.

Keywords. Artificial neural network; pattern recognition; biological neural network.

1. Introduction

Human problem solving is basically a pattern processing problem and not a data processing problem. In any pattern recognition task humans perceive patterns in the input data and manipulate the pattern directly. In this paper we discuss attempts at developing computing models based on artificial neural networks (ANN) to deal with various pattern recognition situations in real life.

Search for new models of computing is motivated by our quest to solve natural (intelligent) tasks by exploiting the developments in computer technology (Marcus & van Dam 1991). The developments in artificial intelligence (AI) appeared promising till a few years ago. But when the AI methods were applied to natural tasks such as in speech, vision and natural language processing, the inadequacies of the methods showed up. Like conventional algorithms, AI methods also need a clear specification of the problem, and mapping of the problem into a form suitable for the methods to be applicable. For example, in order to apply heuristic search methods, one needs to map the problem as a search problem. Likewise, to solve a problem using a rule-based approach, it is necessary to explicitly state the rules governing it. Scientists are hoping that computing models inspired by biological neural networks may provide new directions to solving problems arising in natural tasks. In particular, it is hoped that neural networks would extract the relevant features from input data and perform the pattern recognition task by learning from examples, without explicitly stating the rules for performing the task.

The objective of this tutorial paper is to present an overview of the current approaches based on artificial neural networks for solving various pattern recognition tasks. From the overview it will be evident that the current approaches still fall far short of our expectations, and there is scope for evolving better models inspired by the principles of operation of our biological neural network. This paper is organized as follows: In § 2 we discuss the nature of patterns and pattern recognition tasks that we encounter in our daily life. We make a distinction between pattern and data, and also between understanding and recognition. In this section we also briefly discuss methods available for dealing with pattern recognition tasks, and make a case for new models of computing based on artificial neural networks. The basics of artificial neural networks are presented in § 3, including a brief discussion on the operation of a biological neural network, models of neuron and the neuronal activation and synaptic dynamics. Section 4 deals with the subject matter of this paper, namely, the use of principles of artificial neural networks to solve simple pattern recognition tasks. This section introduces the fundamental neural networks that laid the foundation for developing new architectures. In § 5 we discuss a few architectures for complex pattern recognition tasks. In the final section we discuss several issues that need to be addressed to develop artificial neutral network models for solving practical problems.

2. Patterns and pattern recognition tasks

2.1 *Notion of intelligence*

The current usage of the terms like AI systems, intelligent systems, knowledge-based systems, expert systems etc., are intended to show the urge to build machines that can demonstrate intelligence similar to human beings in performing some simple tasks. In these tasks we look at the performance of a machine and compare it with the performance of a person. We attribute intelligence to the machine if the performances match. But the way the tasks are performed by a machine and by a human being are basically different; the machine performing the task in a step-by-step sequential manner dictated by an algorithm, modified by some known heuristics.

The algorithm and the heuristics have to be derived for a given task. Once derived, they generally remain fixed. Typically, implementation of these tasks requires large number of operations (arithmetic and logical) and also a large amount of memory. The trends in computing clearly demonstrate the machine's ability to handle a large number of operations (Marcus & van Dam 1991).

2.2 *Patterns and data*

However, the mere ability of a machine to perform a large amount of symbolic processing and logical inferencing (as is being done in AI) does not result in intelligent behaviour. The main difference between human and machine intelligence comes from the fact that humans perceive everything as a ***pattern,*** whereas for a machine all are ***data.*** Even in routine data consisting of integer numbers (like telephone numbers, bank account numbers, car numbers), humans tend to see a pattern. Recalling the data is also normally from a stored pattern. If there is no pattern, then it is very difficult for a human being to remember and reproduce the data later. Thus storage and recall operations in humans and machines are performed by different mechanisms. The pattern nature in storage and recall automatically gives robustness and fault tolerance for a human system. Moreover, typically far fewer patterns than the estimated capacity of human memory systems are stored.

Functionally also humans and machines differ in the sense that humans ***understand*** patterns, whereas machines can be said to ***recognize*** patterns in data. In other words, humans can get the whole object in the data even though there may be no clear identification of subpatterns in the data. For example, consider the name of a person written in a handwritten cursive script. Even though individual patterns for each letter may not be evident, the name is understood due to the visual hints provided in the written script. Likewise, speech is understood even though the patterns corresponding to individual sounds may be distorted sometimes to unrecognizable extents. Another major characteristic of a human being is the ability to continuously learn from examples, which is not well understood at all in order to implement it in an algorithmic fashion in a machine.

Human beings are capable of making mental patterns in their biological neural network from input data given in the form of numbers, text, pictures, sounds etc., using their sensory mechanisms of vision, sound, touch, smell and taste. These mental patterns are formed even when the data are noisy, or deformed due to variations such as translation, rotation and scaling. The patterns are also formed from a temporal sequence of data as in the case of speech and motion pictures. Humans have the ability to recall the stored patterns even when the input information is noisy or partial (incomplete) or mixed with information pertaining to other patterns.

2.3 *Pattern recognition tasks*

The inherent differences in information handling by human beings and machines in the form of patterns and data, and in their functions in the form of understanding and recognition have led us to identify and discuss several pattern recognition tasks which human beings are able to perform very naturally and effortlessly, whereas we have no simple algorithms to implement these tasks on a machine. The identification of these tasks below is somewhat influenced by the organization of the artificial neural network models which we will be describing later in this paper.

2.3a ***Pattern association:*** Pattern association problem involves storing a set of patterns or a set of input–output pattern pairs in such a way that when test data are presented, the pattern or pattern pair corresponding to the data is recalled. This is purely a memory function to be performed for patterns and pattern pairs. Typically, it is desirable to recall the correct pattern even though the test data are noisy or incomplete. The problem of storage and recall of patterns is called autoassociation. Since this is a content addressable memory function, the system should display ***accretive*** behaviour, i.e., should recall the stored pattern closest to the given input. It is also necessary to store as many patterns or pattern pairs as possible in a given system.

Printed characters or any set of fixed symbols could be considered as examples of patterns for these tasks. Note that the test patterns are the same as the training patterns, but with some noise added, or some portions missing. In other words, the test data are generated from the same source in an identical manner as the training data.

2.3b ***Pattern mapping:*** In pattern mapping, given a set of input patterns and the corresponding output pattern or class label, the objective is to capture the implicit relationship between the patterns and the output, so that when a test input is given, the corresponding output pattern or the class label is retrieved. Note that the system should perform some kind of ***generalization*** as opposed to ***memorizing*** the information. This can also be viewed as a pattern classification problem belonging to ***supervised*** learning category. Typically, in this case the test patterns belonging to a class are not the same as the training patterns, although they may originate from the same source. Speech spectra of steady vowels generated by a person, or hand-printed characters, could be considered as examples of patterns for pattern mapping problems. Pattern mapping generally displays ***interpolative*** behaviour, whereas pattern classification displays ***accretive*** behaviour.

2.3c ***Pattern grouping:*** In this case, given a set of patterns, the problem is to identify the subset of patterns possessing similar distinct features and group them together. Since the number of groups and the features of each group are not explicitly stated, this problem belongs to the category of ***unsupervised*** learning or pattern clustering. Note that this is possible only when the features are unambiguous as in the case of hand-printed characters or steady vowels. In the pattern mapping problem the patterns for each group are given separately, and the implicit, although distinct, features have to be captured through the mapping. In pattern grouping on the other hand, patterns belonging to several groups are given, and the system has to resolve the groups.

Examples of the patterns for this task could be printed characters or hand-printed characters. In the former case, the grouping can be made based on the data themselves. Moreover, in that case the test data are also generated from an identical source as the training data. For hand-printed characters or steady vowel patterns, the features of the patterns in the data are used for grouping. Therefore in this case the test data are generated from a similar source as the training data, so that only features are preserved and not necessarily the actual data values.

2.3d ***Feature mapping:*** In several patterns the features are not unambiguous. In fact the features vary over a continuum, and hence it is difficult to form groups of patterns having some distinct features. In such cases, it is desirable to display the feature changes in the patterns directly. This again belongs to the unsupervised learning category. In this case what is learnt is the ***feature map*** of a pattern and not the group or class to which the pattern may belong. This occurs, for example, in the speech spectra for vowels in continuous speech. Due to changes in the vocal tract shape for the same vowel occurring in different contexts, the features (formants or resonances of the vocal tract in this case) vary over overlapping regions for different vowels.

2.3e *Pattern variability:* There are many situations when the features in the pattern undergo unspecified distortions each time the pattern is generated by the system. This can be easily seen in the normal handwritten cursive script. Human beings are able to recognize them due to some implicit interrelations among the features, which themselves cannot be articulated precisely. Classification of such patterns falls into the category of pattern variability task.

2.3f *Temporal patterns:* All the tasks discussed so far refer to the features present in a given static pattern. Human beings are able to capture effortlessly the dynamic features present in a sequence of patterns. This is true, for example, in speech where the changes in the resonance characteristics of the vocal tract system (e.g. formant contours) capture the significant information about the speech message. This is also true in any dynamic scene situation. All such situations require handling sequences of static patterns simultaneously, looking for changes in the features in the subpatterns in adjacent pattern pairs.

2.3g *Stability–plasticity dilemma:* In any pattern recognition task the input patterns keep changing. Therefore it is difficult to freeze the categorization task based on a set of patterns used in the training set. If it is frozen, then the system cannot learn the category that a new pattern may suggest. In other words, the system lacks its ***plasticity.*** On the other hand, if the system is allowed to change its categorization continuously, based on new input patterns, it cannot be used for any application such as pattern classification or clustering, as it is not ***stable.*** This is called ***stability–plasticity dilemma*** in pattern recognition.

2.4 *Methods for pattern recognition tasks*

Methods for solving pattern recognition tasks generally assume a sequential model for the pattern recognition process, consisting of pattern environment, sensors to collect data from the environment, feature extraction from the data and association/storage/classification/clustering using the features.

The simplest solution to a pattern recognition problem is to use template matching, where the data of the test pattern are matched point by point with the corresponding data in the reference pattern. Obviously, this can work only for very simple and highly restricted pattern recognition tasks. At the next level of complexity, one can assume a deterministic model for the pattern generation process, and derive the parameters of the model from given data in order to represent the pattern information in the data. Matching test and reference patterns are done at the parametric level. This works well when the model of the generation process is known with reasonable accuracy. One could also assume a stochastic model for the pattern generation process, and derive the parameters of the model from a large set of training patterns. Matching between test and reference patterns can be performed by several statistical methods like likelihood ratio, variance weighted distance, Bayesian classification etc. Other approaches for pattern recognition tasks depend on extracting features from parameters or data. These features may be specific for the task. A pattern is described in terms of features, and pattern matching is done using descriptions of the features. Another method based on descriptions is called syntactic or structural pattern recognition in which a pattern in expressed in terms of primitives suitable for the classes of pattern under study (Schalkoft 1992). Pattern matching is performed by matching the descriptions of the patterns in terms of the primitives. More recently, methods based on the knowledge of the sources generating the patterns are being explored for pattern recognition tasks. These knowledge-based systems express knowledge in the form of rules for generating and perceiving patterns.

The main difficulty in each of the pattern recognition techniques alluded to above is that of choosing an appropriate model for the pattern generating process and estimating the parameters of the model in the case of a model-based approach, ***or*** extraction of features from data/parameters in the case of feature-based methods, ***or*** selecting appropriate primitives in the case of syntactic pattern recognition, ***or*** deriving rules in the case of a knowledge-based approach. It is all the more difficult when the test patterns are noisy and distorted versions of the patterns used in the training process. The ultimate goal is to impart to a machine the pattern recognition capabilities comparable to those of human beings. This goal is difficult to achieve using most of the conventional methods, because, as mentioned earlier, these methods assume a sequential model for the pattern recognition process. On the other hand, the human pattern recognition process is an integrated process involving the use of biological neural processing even from the stage of sensing the environment. Thus the neural processing takes place directly on the data for feature extraction and pattern matching. Moreover, the large size (in terms of number of neurons and interconnections) of the biological neural network and the inherently different mechanism of processing are attributed to our abilities of pattern recognition in spite of variability and noise in the data. Moreover, we are able to deal effortlessly with temporal patterns and also with the so-called stability–plasticity dilemma as well.

It is for these reasons attempts are being made to explore new models of computing, inspired by the structure and function of the biological neural network. Such models for computing are based on artificial neural networks, the basics of which are introduced in the next section.

3. Basics of artificial neural networks

3.1 *Characteristics of biological neural networks*

New models of computing to perform pattern recognition tasks based on our biological neural network are not expected to reach anywhere near the performance of the biological network for several reasons. Firstly, we do not fully understand the operation of a biological neuron and the dynamics of the neural interconnections. Secondly, it is nearly impossible to simulate (i) the number of neurons and their interconnections as it exists in a biological network, and (ii) the dynamics of the network that determines the operation of the network.

The features that make the performance of a biological network superior to even the most sophisticated AI computer system for pattern recognition tasks are the following (Hertz ***et al*** 1991).

(a) Robustness and fault tolerance – The decay of nerve cells does not seem to affect the performance of the network significantly.

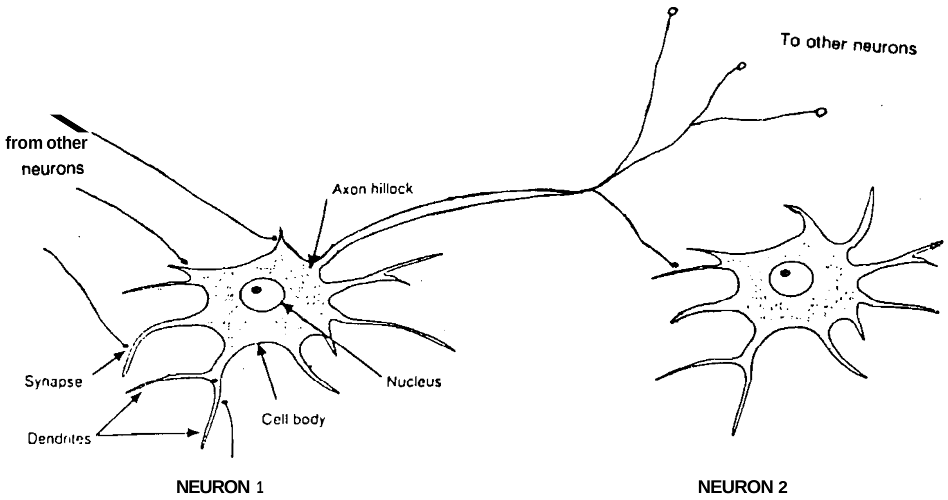


Figure 1. Schematic drawing of a typical neuron or nerve cell. It includes dendrites, the cell body and a single axon. Synapses connect the axons of neurons to various parts of other neurons.

(b) Flexibility – The network automatically adjusts to a new environment without using any preprogrammed instruction set.

(c) Ability to deal with a variety of data situations – The network can deal with information that is fuzzy, probabilistic, noisy or inconsistent.

(d) Collective computation – The network can routinely perform many operations in parallel and also a given task in a distributed manner.

These features are attributed to the structure and function of a biological neural network (Muller & Reinhardt 1990). The fundamental unit of the network is called a neuron or nerve cell. Figure 1 shows a schematic of the structure of a neuron. It consists of a cell body or soma where the cell nucleus is located. Tree-like networks of nerve fibres, called dendrites, are connected to the cell body. Extending from the cell body is a single long fibre, called the axon, which eventually branches into strands and substrands, connecting to many other neurons at the synaptic junctions or synapses. The receiving end of these junctions on other cells can be found both on the dendrites and on the cell bodies themselves. The axon of a typical neuron makes a few thousand synapses with other neurons.

The transmission of a signalfrom one cell to another at a synapse is a complex chemical process, in which specific transmitter substances are released from the sending side of the junction. The effect is to raise or lower the electrical potential inside the body of the receiving cell. If this potential reaches a threshold, electrical activity in the form of short pulses takes place. When this happens, the cell is said to have fired. This electrical activity of fixed strength and duration is sent down the axon.

The dendrites serve as receptors for signals from adjacent neurons, whereas the axon's purpose is the transmission of the generated neural activity to other nerve cells or to muscle fibres. In the first case the term interneuron may be used, whereas the neuron in the latter case is called motor neuron. A third type of neuron, which receives information from muscles or sensory organs, such as the eye or ear, is called a receptor neuron.

Although all neurons operate on the same basic principle, there exist several different types of neurons, distinguished by the size and degree of branching of their dendritic trees, the length of their axons, and other structural details. The complexity of the human central nervous system is due to the vast number of neurons and their mutual connections. Connectivity is characterized by the complementary properties of convergence and divergence. In the human cortex every neuron is estimated to receive converging input on the average from about 10^4 synapses. On the other hand, each cell feeds its output into many hundreds of other neurons. The total number of neurons in the human cortex is estimated to be in the vicinity of 10^{11}, and are distributed in layers over a full depth of cortical tissue at a constant density of about 150,000 neurons per square millimetre. Combined with the average number of synapses per neuron, this yields a total of about 10^{15} synaptic connections in the human brain, the majority of which develop during the first few months after birth. The study of the properties of complex systems built of simple, identical units, may lead to an understanding of the mode of operation of the brain in its various functions, although we are still far from it.

The simplified schematic and uniform connectionist units offer a surprisingly rich structure when assembled in a closely interconnected network. We shall call such a network an ***artificial neural network.*** Since artificial neural networks are implemented on computers, it is worth comparing the processing capabilities of computers with that of the biological neural networks (Simpson 1990).

Neural networks are slow in processing information. The cycle time corresponding to execution of one step of a program in a computer is in the range of a few nanoseconds, whereas the cycle time corresponding to a neural event prompted by an external stimulus, is in the millisecondsrange. Thus computers process information a million times faster.

Neural networks perform massively parallel operations. Most programs operate in a serial mode, one instruction after another, in a conventional computer, whereas the brain operates with massively parallel programs that have comparatively fewer steps.

Neural networks have large numbers of computing elements, and the computing is not restricted to within neurons. The conventional computer typically has one central processing unit where all the computing takes place.

Neural networks store information in the strengths of the interconnections. In a computer, information is stored in the memory which is addressed by its location. New information is added by adjusting the interconnection strengths without completely destroying the old information, whereas in a computer the information is strictly replaceable.

Neural networks distribute the encoded information throughout the network, and hence they exhibit fault tolerance. In contrast, computers are inherently not fault tolerant, in the sense that information corrupted in the memory cannot be retrieved.

There is no central control in processing information in the brain. Thus there in no specific control mechanism external to the computing task. In a computer, on the other hand, there is a control unit which monitors all the activities of computing.

While the superiority of the human information processing system over the conventional computer for pattern recognition tasks stems from the basic structure and operation of the biological neural network, it is possible to realize some of the features of the human system using an artificial neural network consisting of basic computing elements. In particular, it is possible to show that such a network exhibits parallel and distributed processing capability. In addition, information can be stored in a distributed manner in the connection strengths so as to achieve fault tolerance.

3.2 *Artificial neural networks – terminology*

3.2a ***Processing unit:*** We can consider an artificial neural network (ANN) as a highly simplified model of the structure of the biological neural network. An ANN consists of interconnected ***processing units.*** The general model of a processing unit consists of a summing part followed by an output part. The summing part receives ***n*** input values, weighs each value, and performs a weighted sum. The weighted sum is called the ***activation value.*** The sign of the weight for each input determines whether the input is ***excitatory*** (positive weight) or ***inhibitory*** (negative weight). The inputs could be discrete or continuous data values, and likewise the outputs also could be discrete or continuous. The input and output may also be viewed as deterministic or stochastic or fuzzy, depending on the nature of the problem and its solution.

3.2b ***Interconnections:*** In an artificial neural network several processing units are interconnected according to some topology to accomplish a pattern recognition task. Therefore the inputs to a processing unit may come from outputs of other processing units, and/or from an external source. The output of each unit may be given to several units including itself. The amount of the output of one unit received by another unit depends on the strength of the connection between the units, and it is reflected in the ***weight*** value associated with the connecting link. If there are N units in a given ANN then at any instant of time each unit will have a unique activation value and a unique output value. The set of the N activation values of the network defines the ***activation state*** of the network at that instant. Likewise, the set of the N output values of the network define the ***output state*** of the network at that instant. Depending on the discrete or continuous nature of the activation and output values, the state of the network can be described by a point in a discrete or continuous N-dimensional space.

3.2c ***Operations:*** In operation, each unit of an ANN receives inputs from other connected units and/or from an external source. A weighted sum of the inputs is computed at a given instant of time. The resulting activation value determines the actual output from the ***output function*** unit, **i.e.,** the output state of the unit. The output values and other external inputs in turn determine the activation and output states of the other units. The activation values of the units (activation state) of the network as a function of time are referred to as ***activation dynamics.*** The activation dynamics also determine the dynamics of the output state of the network. The set of all activation states defines the ***state space*** of the network. The set of all output states defines the ***output*** or ***signal state space*** of the network. Activation dynamics determines the trajectory of the path of the states in the state space of the network.

For a given network, defined by the units and their interconnections with appropriate weights, the activation states refer to the ***short term memory*** function of the network. Generally the activation dynamics is followed to ***recall*** a pattern stored in a network.

In order to store a pattern in a network, it is necessary to adjust the weights of the network. The sets of all weight values (corresponding to strengths of all connecting links of an ANN) defines the ***weight space.*** If the weights are changing, then the set of weight values as a function of time defines the synaptic dynamics of the network. Synaptic dynamics is followed to adjust the weights in order to store given patterns in the network. The process of adjusting the weights is referred to as learning. Once the learning process is completed, the final set of weight values corresponds to the long term memory function of the network. The procedure to incrementally update each of the weights is called a learning law or learning algorithm.

3.2d Update: In implementation, there are several options available for both activation and synaptic dynamics. In particular, the updating of the output states of all units could be performed synchronously. In this case, the activation values of all units are computed at the same time assuming a given output state throughout. From these activation values the new output state of the network is derived. In an asynchronous update, on the other hand, each unit is updated sequentially, taking the current output state of the network into account each time. For each unit, the output state can be determined from the activation value either deterministically or stochastically.

In practice, the activation dynamics, including the update, is much more complex in a biological neural network. The ANN models along with the equations governing the activation and synaptic dynamics are developed according to the complexity of the pattern recognition task to be handled.

3.3 Models of neurons

In this section we will consider three classical models for an artificial neuron or processing unit.

3.3a *McCulloch–Pitts* model: In the **McCulloch–Pitts** (MP) model (figure 2) the activation (x) is given by a weighted sum of its n-input signal values $\{a_i\}$ and a bias term (8). The activation could have an additional absolute inhibition term, which can prevent excitation of the neuron. The output signal (s) is typically a nonlinear function of the activation value. Three common nonlinear functions (binary, ramp and sigmoid) are shown in figure 3, although the binary function was used in the original MP model. The following equations describe the operation of an MP model:

activation: $$x = \sum_{i=1}^{n} w_i a_i - \theta - [\text{inhibition}],$$

output signal: $$s = f(x).$$

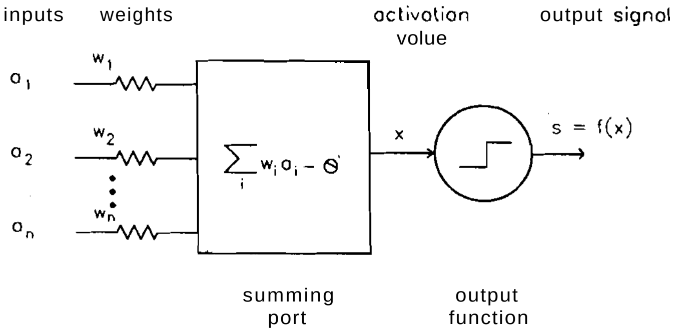


Figure 2. The **McCulloch–Pitts** model of a neuron.

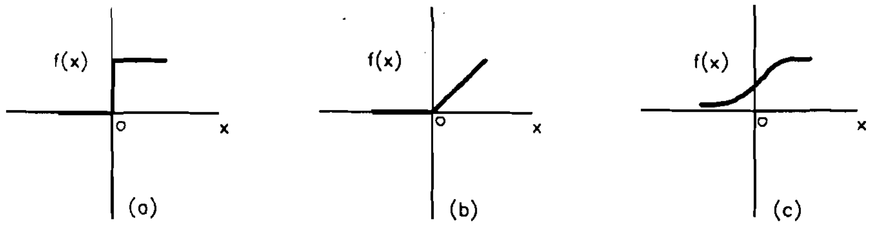


Figure 3. Some nonlinear functions. (a) Binary, (b) ramp and (c) sigmoid.

In this model the weights w_i are constant. That means there is no learning. Networks consisting of MP neurons with binary (on–off) output signals can be configured to perform several logical functions (McCulloch & Pitts 1943).

3.3b Perceptron: Rosenblatt's perceptron model (figure 4) for an artificial neuron consists of outputs from sensory units to a fixed set of association units, the outputs of which are fed to an MP neuron (Rosenblatt 1958). The association units perform predetermined manipulations on their inputs. The main deviation from the MP model is that here learning (i.e., adjustment of weights) is incorporated in the operation of the unit. The target output (b) is compared with the actual binary output (s) and the error is used to adjust the weights (Rosenblatt 1962). The following equations describe the operation of'the perceptron model of a neuron.

activation: $x = \sum_{i=1}^{n} w_i a_i - \theta,$

output signal: $s = f(x),$

error: $\delta = b - s,$

weight update: $\frac{dw_i}{dt} = \eta \delta a_i,$

where η is called learning rate parameter.

There is the perceptron learning law which gives a step-bystep procedure for adjusting the weights. Whether the adjustment converges or not depends on the

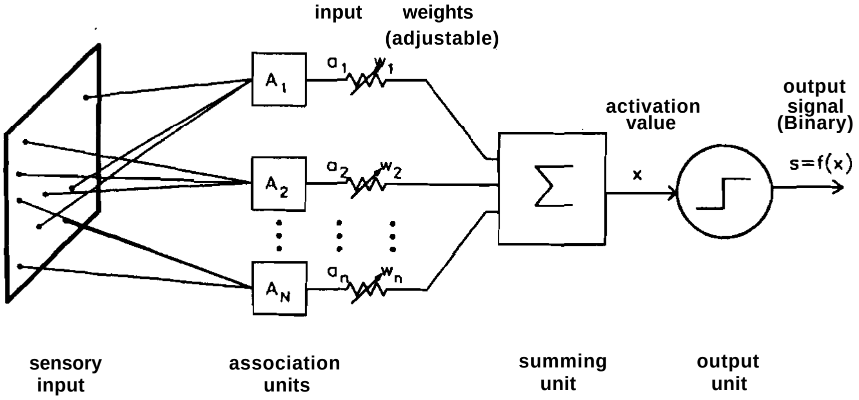


Figure 4. Rosenblatt's model of a neuron.

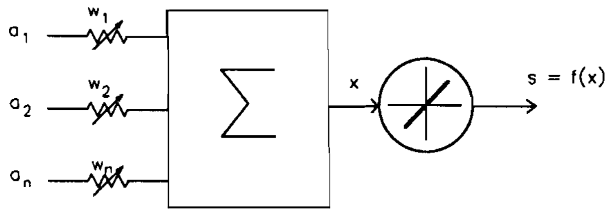


Figure 5. Widrow's adaline model of a neuron.

nature of the desired input–output pair to be represented by the model. The perceptron convergence theorem (Rosenblatt 1962) enables us to determine whether a given pattern pair is representable or not. If the weight values converge, then the corresponding problem is said to be representable by the perceptron network.

3.3c Adaline: The main distinction between Rosenblatt's perceptron model and Widrow's adaline model (figure 5) is that in the adaline model the analog activation value (x) is compared with the target output (b). In other words, the output is a linear function of the activation value (x). The equations that describe the operation of an adaline are as follows (Widrow & Hoff 1960):

activation: $$x = \sum_{i=1}^{n} w_i a_i - \theta,$$

output signal: $$s = f(x) = x,$$

error: $$\delta = b - s = b - x,$$

weight update: $$dw_i/dt = \eta \delta a_i.$$

This rule minimizes the mean squared error δ^2, averaged over all inputs. Hence it is called the least mean squared (LMS) error learning law. The law is derived using the negative gradient of the error surface in the weight space. Hence it is also called a gradient descent algorithm.

3.4 Topology

Artificial neural networks are useful only when the processing units are organized in a suitable manner to accomplish a given pattern recognition task. This section presents a few basic structures which will assist in evolving new architectures. The arrangement of the processing units, connections, and pattern input/output is referred to as topology (Simpson 1992, pp. 3–24).

Artificial neural networks are normally organized into layers of processing units. Connections can be made either from units of one layer to units of another (interlayer connections) or from the units within the layer (intralayer connections) or both inter and intralayer connections. Further, the connections among the layers and among the units within a layer can be organized either in a feedforward manner or in a feedback manner. In a feedback network the same processing unit may be visited more than once.

We will discuss a few basic structures which form building blocks for complex neural network architectures. Let us consider two layers F_1 and F_2 with N and M processing units, respectively. By providing connections to the jth unit in F_2 from all the units in F_1, as shown in figures 6a and b, we get two network structures **instar** and **outstar**, which have fan-in and fan-out geometries, respectively. The units in the

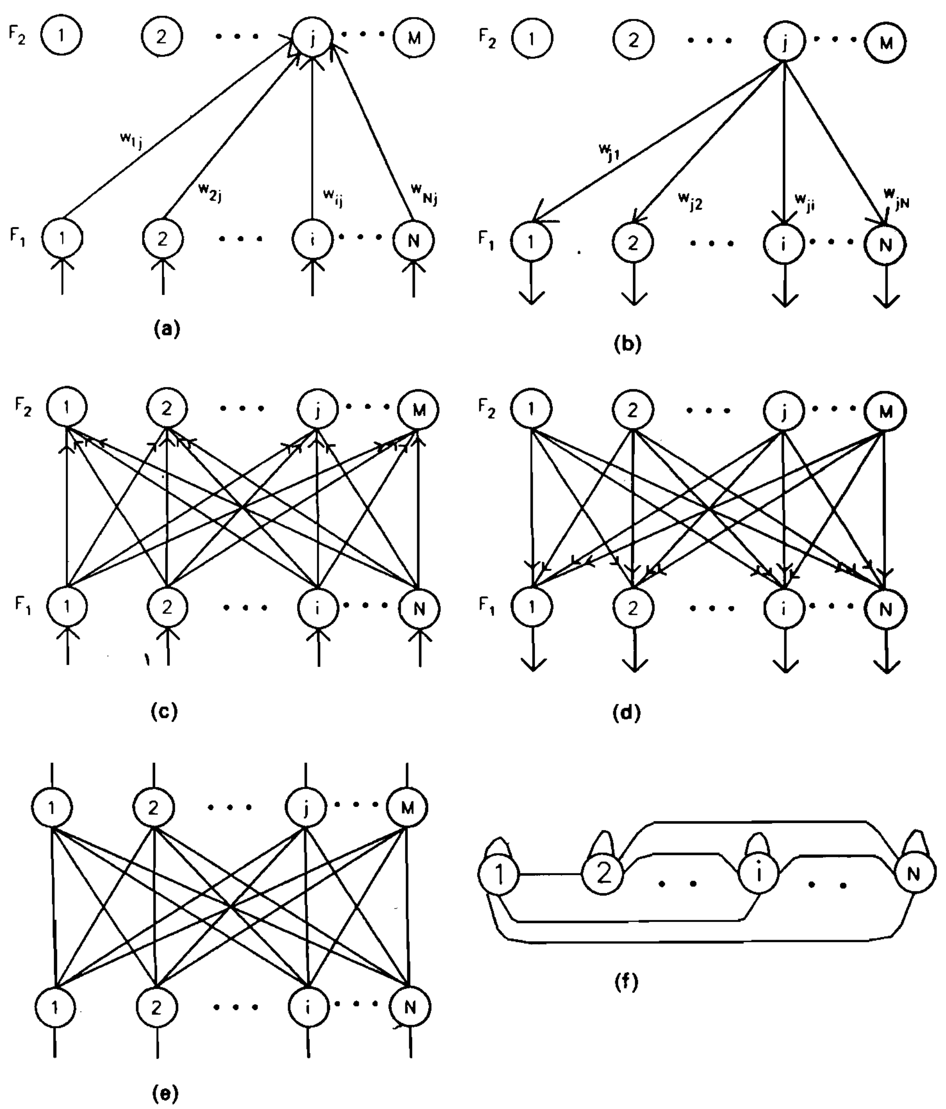


Figure 6. Some basic structures of the Artificial Neural Networks. (a) Instar, (b) outstar, (c) group of instars, (d) group of outstars, (e) bidirectional associative memory, and (f) autoassociative memory.

F_1 layer are linear units, so that for each unit i in this layer the input (a,) = activation (x_i) = output signal (s,). In **instar,** during learning, the weight vector $\mathbf{w}_j(w_{j1}, w_{j2},..., w_{jN})$ is adjusted so as to approach the given input vector a at F_1 layer. Therefore whenever the input is given to F_1, then the jth unit of F_2 will be activated to the maximum extent. Thus the operation of the **instar** can be viewed as content addressing the memory. In the case of the **outstar,** during learning, the weight vector for the connections from the jth unit in F_2 approaches the activity pattern in F_1 when input vector a is present at F_1. During recall, whenever the unit j is activated, the signal pattern ($s_j w_{1j}$, $s_j w_{1j}$,...$s_j w_{Nj}$) will be transmitted to F_1, which then produces the original activity pattern corresponding to the input vector a, although the input is absent. Thus the operation of the **outstar** can be viewed as memory addressing the contents.

When all the connections from units in F, and F, are made as in figure **6c**, then we obtain a heteroassociation network. This network can be viewed as a group of **instars**, if the flow is from F_1 to F,. On the other hand, if the flow is from F, to F,, then the network can be viewed as a group of **outstars** (figure 6d).

When the flow is bidirectional, and the weights are symmetric $w_{ij} = w_{ji}$, then we get a bidirectional associative memory (figure **6e**), where either of the layers can be used as **input/output**.

If the two layers F_1 and F, coincide, then we obtain an autoassociative memory in which each unit is connected to every other unit and to itself (figure 6f).

3.5 Activation and synaptic dynamics

Artificial neural networks can be considered as trainable nonlinear dynamical systems (Kosko 1972). For a network consisting of N processing units, the activation state of the network at any given instant corresponds to a point in the N-dimensional state space. The dynamics of the neural network traces a trajectory in the state space. The trajectory begins with a point in the state space representing a computational problem and ends at a point in the state space representing a computational solution. Most of the trajectory corresponds to the transient behaviour of computations. The trajectory ends at an equilibrium state of the system in the normal course. An equilibrium state is one at which small perturbations around it due to neuronal dynamics will not perturb the state.

Neuronal dynamics consists of two parts: one corresponding to the dynamics of activation states and the other corresponding to the dynamics of synaptic weights. The activation dynamics determines the time evolution of the neuronal activations, and it is described by a system of first order differential equations. The equations governing the dynamics are described in terms of the first derivative of the activation state, **i.e.**, $\mathrm{d}x_i/\mathrm{d}t$. Likewise synaptic dynamics determines the changes in the synaptic weights. The equations governing the dynamics are described in terms of the first derivative of the synaptic weights, **i.e.**, $\mathrm{d}w_{ij}/\mathrm{d}t$, where w_{ij} is the strength of the connecting link'from the jth unit to the ith unit. Synaptic weights change gradually, whereas the neuronal activations fluctuate rapidly. Therefore, while computing the activation dynamics, the synaptic weights are assumed to be constant. The synaptic dynamics dictates the learning process. The short term memory (STM) in neural networks is modelled by the activation state of the network. The long term memory (LTM) corresponds to the encoded pattern information in the synaptic weights due to learning.

3.5a Models of activation dynamics: Different models are proposed for the activation dynamics, the most common ones among them are the additive and shunting activation models. The additive activation model is given by the equation for the rate of change of the activation of the ith unit as (Grossberg 1988; Carpenter 1989).

$$\mathrm{d}x_i/\mathrm{d}t = -x_i + \sum[\text{excitatory inputs}] - \sum[\textbf{inhibitory}\text{ inputs}].$$

In this equation the first term on the right hand side contributes to a passive decay term. The net excitatory and inhibitory inputs are contributed by signals from other units appropriately weighted by the synaptic strengths and by the externally applied inputs.

In the steady state there will not be any change in activation. That is $\mathrm{d}x_i/\mathrm{d}t = 0$. In such a case the activation value is given by the net excitatory and inhibitory inputs. That is

$$x_i = \sum[\text{excitatory inputs}] - \sum[\text{inhibitory inputs}].$$

For a specific case x_i can be written as

$$x_i = \sum_j w_{ij} s_j - \theta_i + I_i.$$

The sign of w_{ij} determines whether the contribution is excitatory or inhibitory. θ_i is a fixed bias term for the unit, and it becomes the resting value in the absence of all inputs. I_i is the net external input to the unit i. The sign of I_i determines whether it is excitatory or inhibitory.

An important generalization of the additive model is the shunting activation model given by the equation (Grossberg **1988**),

$$\mathrm{d}x_i/\mathrm{d}t = -x_i + (\mathrm{A} - x_i)\sum[\text{excitatory inputs}] - (\mathrm{B} + x_i)\sum[\text{inhibitory inputs}],$$

where the activity x_i remains bounded in the range $(-\mathrm{B}, \mathrm{A})$, and it decays to the resting level **0** in the absence of all inputs. In this model the excitatory inputs drive the activity towards a finite maximum A, and the inhibitory inputs drive the activity towards a finite minimum $-\mathrm{B}$. The shunting model represents a special case of Hodgkin–Huxley membrane equations to describe the physiology of single nerve cell dynamics (Hodgkin & Huxley 1952).

The activation models considered so far are called deterministic models. In practice, the **input/output** patterns and the activation values can be considered as samples of a random process, and the output signal of each unit may be a random function of the unit's activation value. In such a case the network activation state can be viewed as a vector stochastic process. Each unit in turn behaves as a scalar stochastic process (Kosko 1992).

3.5b Models *of* synaptic dynamics: Synaptic dynamics is described in terms of expressions for the first derivative of the weights. They are called learning equations (Kosko 1992). Typical (basic) learning involves adjustment of the weight vector such that

$$\Delta \mathbf{w}_i(t) = \eta g[\mathbf{w}_i(t), \mathbf{a}(t), b_i(t)]\,\mathbf{a}(t),$$
$$\mathbf{w}_i(t + 1) = \mathbf{w}_i(t) + \Delta \mathbf{w}_i(t),$$

where

η = learning rate parameter,
$\mathbf{w}_i = [w_{i1}, w_{i2} \ldots w_{iN}]^T$ weight vector with components w„,
w_{ij} = weight connecting the jth input unit to the ith processing unit,
a = input vector with components a,, i = **1, 2** . . . N,
b = desired output vector with components b„ i = **1, 2, . . .** M.
Input units are assumed linear. Hence a = **x** (unit **activation) = s(unit** output).
Output units are in general nonlinear. Hence $s_i = \mathrm{f}\,(\mathrm{w}_i{}^{\mathrm{T}}\mathrm{a})$.

The function g may be viewed as a learning function that depends on the type of learning adopted.

Continuous time learning can be expressed

$$\frac{d\mathbf{w}_i(t)}{dt} = \eta g[\mathbf{w}_i(t), \mathbf{a}(t), b_i(t)]\mathbf{a}(t).$$

In discrete time learning, at the kth step the new weight is given by

$$\mathbf{w}_i^{k+1} = \mathbf{w}_i^k + \eta g[\mathbf{w}_i^k, a^k, b_i^k]a^k.$$

There are different methods for implementing the synaptic dynamics. These methods are called learning laws. A few common discrete time learning laws are given below (Zurada 1992).

3.5c *Hebb's law (Hebb 1949):*

Here $g(.) = f(w_i^T a)$, where f is the output function. Therefore

$$\Delta w_{ij} = \eta f(\mathbf{w}_i{}^T\mathbf{a})a_j$$
$$= \eta s_i a_j, \quad \text{for} \quad j = 1,2 \ldots N$$

This law requires weight initialization $\mathbf{w}_i \approx \mathbf{0}$ prior to learning.

3.5d *Perceptron learning law (Rosenblatt 1962):*

Here $g(.) = b_i - s_i = b_i - sgn(w^r a)$. Therefore

$$\Delta w_{ij} = \eta[b_i - sgn(w^T a)]aj, \quad \text{for} \quad j = 1,2 \ldots N.$$

This rule is applicable for bipolar output function. The weights can be initialized to any values prior to learning.

3.5e *Delta learning law:*

Here $g(.) = [b_i - f(\mathbf{w}_i{}^T\mathbf{a})] f'(\mathbf{w}_i{}^T\mathbf{a})$. This is obtained by setting

$$\Delta\mathbf{w}_i = -\eta\nabla E$$

where $-VE$ is the negative gradient of the error $E = \frac{1}{2}[b_i - f(\mathbf{w}_i{}^T\mathbf{a})]^2$. Therefore

$$\Delta w_{ij} = \eta(b_i - s_i) f'(\mathbf{w}_i{}^T\mathbf{a})a_j, \quad \text{for} \quad j = 1,2 \ldots N$$

Here f (.) is a continuous function. The weights may be initialized to any values.

3.5f *Widro–Hoff* LMS *learning law (**Widrow** & Hoff 1960):*

Here $g() = b_i - \mathbf{w}_i{}^T\mathbf{a}$. Therefore

$$Aw_{ij} = \eta(b_i - \mathbf{w}_i{}^T\mathbf{a})a_j, \quad \text{for} \quad j = 1,2 \ldots N.$$

This is a special case of the delta learning law where the output function is assumed to be linear, i.e., f $(w_i^T a) = w_i^T a$. The weights may be initialized to any values.

3.5g *Correlation learning law:*

$$\Delta w_{ij} = \eta b_i a_j, \quad \text{for} \quad j = 1, 2 \ldots \mathrm{N}.$$

This is applicable for binary output units. This is a special case of Hebbian learning with output signal $(s,)$= desired signal (b_i). The weights are initialized to zero prior to learning.

3.5h *Instar (winner-take-all) learning law (Grossberg 1982):*

$$\Delta w_{mj} = \eta (a_j - w_{mj}), \quad \text{for} \quad j = 1, 2, \ldots \mathrm{N},$$

where $\mathbf{w}_\mathbf{m}^T \mathbf{a} = \max_i (\mathbf{w_i}{}^T \mathbf{a})$. Here the weights are initialized to random values prior to learning and their lengths are normalized during learning.

3.5i *Outstar learning law (Grossberg 1982):*

$$\Delta w_{kj} = \eta (b_k - w_{kj}), \quad \text{for} \quad k = 1, 2 \ldots \mathrm{K}$$

where b is the desired response from the layer of K neurons. The weights are initialized to zero before learning.

There are several learning laws in use, and new laws are being developed to suit a given application and architecture. Some of these will be discussed in the appropriate sections later. But there are some general categories that these laws fall into, based on the characteristics they are expected to possess for different applications. In first place, the learning or weight changes could be ***supervised*** or ***unsupervised.*** In supervised learning the weight changes are determined by the difference between the desired output and the actual output. Some of the supervised learning laws are: error correction learning or delta rule, stochastic learning, and hardwired systems (Simpson 1992, pp. 3–24). Supervised learning may be used for ***structural learning*** or for ***temporal learning.*** Structural learning is concerned with capturing in the weights the relationship between a given input-output pattern pair. Temporal learning is concerned with capturing in the weights the relationship between neighbouring patterns in a sequence of patterns.

Unsupervised learning discovers features in a given set of patterns and organizes the patterns accordingly. There is no externally specified desired output as in the case of supervised learning. Examples of unsupervised learning laws are: Hebbian learning, differential Hebbian learning, principle component learning and competitive learning (Simpson 1992, pp. 3–24). Unsupervised learning uses mostly local information to update the weights. The local information consists of signal or activation values of the units at either end of the connection for which the weight update is being made.

Learning methods can be grouped into ***off-line*** and ***on-line.*** In off-line learning all the given patterns are used, may be several times if needed, to adjust the weights. Most error correction learning laws belong to the off-line category. In on-line learning each new pattern or set of patterns can be incorporated into the network without any loss of the prior stored information. Thus an on-line learning allows the neural network to add new information continuously. An off-line learning provides superior solutions because information is extracted when all the training patterns are available, whereas an on-line learning updates only the available information of the past patterns in the form of weights.

In practice, the training patterns can be considered as samples of random processes. Learning laws could take into account the changes in the random process reflected through the samples patterns. Thus one could define stochastic versions of the deterministic learning laws described so far. The random learning laws are expressed as first order stochastic differential equations. For example, the random signal Hebbian learning law relates random processes as (Kosko 1992)

$$\mathrm{d}w_{ij}/\mathrm{d}t = -w_{ij} + \eta s_i s_j + n_{ij},$$

where the output random process $\{s_i\}$ is a result of the signal random process $\{s_j\}$, which in turn may be a result of another activation random process caused by the input process. $\{n_{ij}\}$ can be assumed to be a zero-mean Gaussian white noise process.

In supervised learning one can derive a stochastic approximation to the learning law using the following argument: Given a set of L random sample's, each sample consisting of the pattern pairs (a,, b,), a supervised learning attempts to minimize an unknown error functional $E[\delta_l]$, where 6, is the error between the desired output and the actual output signal. The gradient of $-E[\delta_l]$ points in the direction of steepest descent on the unknown expected error surface. Since the joint probability density function of the input/output pattern pairs is not known, only the error 6, is used as an estimate of $E[\delta_l]$. Since 6, is also a random process, for each iteration in a discrete stochastic gradient descent algorithm, the weight update at the (k + 1)th iteration is given by (Kosko 1992)

$$w_{ij}^{k+1} = w_{ij}^{k} - \eta \delta_{li}^{k} a_{lj},$$

where $\delta_{li}^k = b_{li} - s_i^k$. Since the given data are sample functions of a random process, the corresponding weights at each iteration are also random.

Synaptic equilibrium in the deterministic signal Hebbian law occurs in the steady state when the weights stop changing. That is,

$$\mathrm{d}w_{ij}/\mathrm{d}t = 0, \quad \text{for all} \quad i, j.$$

In the stochastic case the synaptic weights reach a stochastic equilibrium when the changes in the weights are contributed by only the random noise. That is, at stochastic equilibrium, the expectation or ensemble average of the change in weights is given by (Kosko 1990)

$$E[(\mathrm{d}w_{ij}/\mathrm{d}t)^2] = \sigma_{ij}^2,$$

where σ_{ij}^2 is the variance of the noise process n_{ij}.

3.5j Stability and *convergence*: So far the activation and synaptic dynamics equations are described in terms of first-order differential equations which are continuous time equations. Discrete time versions of these equations are convenient for implementation of the network dynamics on a digital computer. In discrete time implementation the activation state of each unit at each stage is computed in terms of the state of the network in the previous stage. The state update at each stage could be made asynchronously, i.e. each unit is updated using the new updated state, or synchronously, i.e., all the units are updated using the same previous state.

The implications of these implementations are on the stability of the equilibrium activation states of a feedback neural network, and on the convergence of the synaptic weights while minimizing the error between the desired output and the actual output during learning. In general, there are no standard methods to determine whether network activation dynamics or synaptic dynamics leads to stability or convergence, respectively, or not (Kosko 1992; **Simpson** 1992, pp. 3–24).

3.5k Neural network recall: During learning, the weights are adjusted to store the information in a given pattern or a pattern pair. However, during performance, the weight changes are suppressed, and the input to the network determines the output activation x_j or signal values s_j. This operation is called recall of stored information. The recall techniques are different for feedforward and feedback networks.

The simplest feedforward network uses the following equation to compute the output signal from the input data vector a to the input layer *F,:*

$$s_i = f_i\left(\sum_j w_{ij} a_j\right),$$

where f_i is the output function of the ith unit in the output layer *F,.* Here the units in the input layer *F ,* are assumed to be linear.

A recall equation for a network with feedback connections is given by (**Simpson** 1992, **pp.** 3–24)

$$x_i(t+1) = (1-\alpha)x_i(t) + \beta \sum_{j=1}^{N} f_j(x_j(t))w_{ij} + a_j,$$

where $x_i(t+1)$ is the activation value of the ith unit in a single layer neural network at time (t + 1), f_j is the nonlinear output function of the jth unit, a is a positive constant that regulates the amount of decay the unit has during the update interval, β is a positive constant that regulates the amount of feedback the other units provide to the ith unit, and a_i is the external input to the ith unit. In general, stability is the main issue in feedback networks. If the network reaches a stable state in a finite number of iterations, then the resulting output signals represent the nearest neighbour stored pattern of the system for the approximate input pattern a.

Cohen & Grossberg (1983) showed that for a wide class of neural networks with certain constraints, the network with fixed weights reaches a stable state in a finite period of time for any initial condition. Later Kosko showed that a neural network could learn and recall at the same time, and yet remain stable (Kosko 1990).

The response of a network due to recall could be the nearest neighbour or interpolative. In the nearest neighbour case, the stored pattern closest to the input pattern is recalled. This typically happens in the feedforward pattern classification or feedback pattern matching networks. In the interpolative case, the recalled pattern is a combination of the outputs corresponding to the input training patterns nearest to the given input test pattern. This happens in the feedforward pattern mapping networks.

4. Functional units of ANN for pattern recognition tasks

So far we have considered issues in pattern recognition and introduced basics of artificial neural networks. In this section we discuss some functional units of artificial neural networks that are useful to solve simple pattern recognition tasks. In particular,

1. Feedforward ANN
   (a) Pattern association
   (b) Pattern classification
   (c) Pattern mapping/classification

2. Feedback ANN
   (a) Autoassociation
   (b) Pattern storage (LTM)
   (c) Pattern environment storage (LTM)

3. Feedforward and Feedback ANN
   (a) Pattern storage (STM)
   (b) Pattern clustering
   (c) Feature map

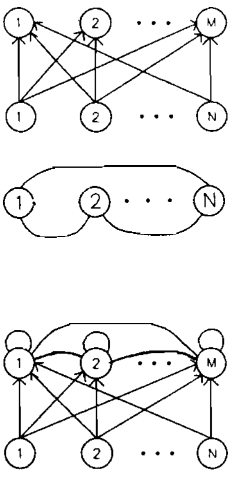


Figure 7. Summary of ANN for pattern recognition problems.

we discuss artificial neural networks for the following pattern recognition problem and for various special cases of the problem.

PROBLEM

Design a neural network to associate the pattern pairs $(a,,\mathbf{b}_1)$, $(a,, b,), \ldots (\mathbf{a}_L, \mathbf{b}_L)$, where $a, = (a,,,a,,, \ldots a,,)$ and $b, = (b,,,b,,, \ldots b,,)$ are N and M dimensional vectors, respectively.

Figure 7 shows the organization of the networks and the pattern recognition tasks to be discussed in this section. We consider three types of ANN: Feedforward, feedback and a combination of both. We begin discussion of each network with only a minimal structure, and study their capabilities and limitations. To start with, the feedforward network consists of two layers of processing units, one layer with linear units for receiving the external input, and the other layer for delivering the output. A minimal feedback network consists of a set of processing units, each connected to all other units. A combination network consists of an input layer of linear units feeding to the output layer of units in a feedforward manner, and a feedback connection among the units in the output layer, including self feedback. We consider each one of these networks in some detail.

4.1 *Pattern recognition tasks by feedforward* ANN *(figure 8)*

4.1a ***Pattern association:*** The objective is to design a linear network that can capture the association in the pairs of vectors $(a,,b,)$, 1= 1,2...L, through a set of weights to be determined by a learning or training law. The input data used in training are typically generated synthetically, like machine printed characters. The input data used for recall may be corrupted by external noise.

The network consists of a set of weights connecting the two layers of processing units, the output function of each unit being linear. Such a network is called a linear associator network. Due to linearity of the output function of each unit, the activation values and the output signals of the units in the input layer are same as the input

Pattern association

- *Arch:* Two layers, linear. processing unit, single set of weights
- *Learning:* Hebb (orthogonal) rule, Delta (linearly independent) rule
- *Recall:* Direct
- *Limitation:* Linear independence, # patterns restricted to dimensionality
- *To overcome:* Nonlinear processing unit, becomes a pattern classification problem

Pattern classification

- *Arch:* Two layers, nonlinear processing units, geometrical interpretation
- *Learning:* Delta rule
- *Recall:* Direct
- *Limitation:* Linearly separable functions, hard problems
- *To overcome:* More layers, hard learning problems

Pattern mapping/classification

- *Arch:* Multilayer (hidden), nonlinear processing units, geometric interpretation
- *Learning:* Generalized delta rule – backpropagation
- *Recall:* Direct
- *Limitation:* Slow learning
- *To overcome:* More complex architectures

Figure 8. Pattern recognition tasks by feedforward ANN.

data values. The activation value of the ith unit in the output layer is given by

$$y_i = \sum_{j=1}^{N} w_{ij} a_{lj}.$$

The output of the ith unit is the same as its activation value y_i, since the output function of the unit is linear. The objective is to determine a set of weights w_{ij} in such a way that the actual output b'_{li} is equal to the desired output b_{li} for all the L pattern pairs.

If the input L pattern vectors *(a,}* are all orthogonal, then it is possible to use Hebb's learning law to determine the optimal weights of the network (Hecht-Nielsen **1990).** Note that a learning law enables updating of weights as patterns are applied one by one to the network. The optimality of the weights is determined by minimizing the mean squared error between the desired and the actual output values. The optimal weights after l pattern pairs are fed to the network are given by

$$w_{ij}^{l} = w_{ij}^{l-1} + b_{li} a_{lj}, \quad w_{ij}^{0} = 0.$$

The final optimal weights for pattern association task are given by

$$w_{ij} = w_{ij}^{L}.$$

If the input vectors *(a,}* are only linearly independent, but not necessarily orthogonal, then the optimal weights that minimize the mean squared error can be obtained using the LMS learning law **(Widrow &** Hoff **1960;** Hecht-Nielsen **1990).**

Once the network is trained, for any given input pattern *a,,* the associated pattern **b,** can be recalled using the equations

$$y_i = \sum_{j=1}^{N} w_{ij} a_{lj} \quad \text{and} \quad b_{li} = y_i.$$

When noisy input patterns are used during recall, i.e., {a,,), then the recalled pattern $\{b'_{li}\}$ will also be noisy. Since the given set of input pattern {a,), $l = 1 \ldots$ L, is assumed to be linearly independent, the number of patterns in the input set is limited to the dimensionality of the input vector, namely, N. Therefore, it is not possible to store more than N pattern pairs in a linear associative network. If the number of input pattern are more than its dimension (N), or if the input set (even for $L < N$) are not linearly independent, then the resulting weight vectors are not optimal any more. In such a case the recall of the associative pattern for a given input pattern may not be correct always.

Even if the input patterns are linearly independent and optimal weights are used, the recall may be in error if a noisy input pattern is presented to recall the associated pattern (Murakami & Aibara 1987).

In practice, linear independence is too severe a restriction to satisfy. Moreover the number of input patterns may far exceed the dimensionality of the input pattern space. It is possible to overcome these limitations by using nonlinear output functions in the processing units of the feedforward ANN. Once the restriction on the number of input patterns is removed, then the problem becomes a pattern classification problem, which we will discuss in the next section.

4.1b Pattern classification: In an N-dimensional space if a set of points could be considered as input patterns without restriction on their number, and if an output pattern, not necessarily distinct, is assigned to each of the input patterns, then the number of distinct output patterns can be viewed as distinct classes or class labels for the input patterns. Since there is no restriction on the type and number of input patterns, the input–output pattern pairs (a,, b,), $1 = 1, 2, \ldots$ L in this case can be considered as a training set for a pattern classification problem. Typically for pattern classification problems the output patterns are points in a discrete (normally binary) M-dimensional space. The input patterns are usually from natural sources like speech and hand-printed characters. The input patterns may be corrupted by external noise at the time of recall.

A two-layer network with nonlinear (threshold or hardlimiting) output function for the units in the output layer, can be used to perform the task of pattern classification. This may also be identified as a single layer perceptron network (Rosenblatt 1962). The network can be trained (i.e., weights can be adjusted) for the given set of input–output patterns using a delta rule.

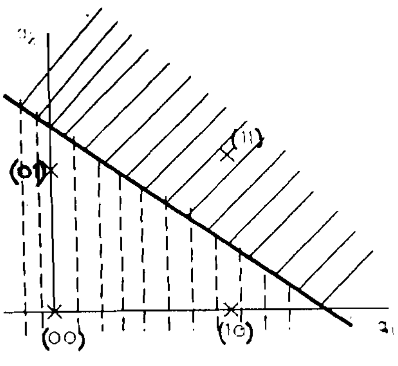
The corresponding learning is also called perceptron learning low (Rosenblatt 1962; Minsky & Papert 1988). The training patterns are applied several times, if needed, until the weights do not change appreciably. But there is no guarantee that the weights will converge to some stable values. Convergence of the weights depends on whether the problem specified by the input–output pattern pairs is representable or not by a network of this type. For all representable problems the learning law converges.

During recall, a pattern generated from one of the same sources is given as input. By direct computation of the weighted sum of the input, the network determines the pattern class to which the input belongs. The network thus exhibits accretive behaviour. Even when the input pattern is noisy, the output class may still be correct, provided the noise has not significantly altered the input pattern.

The unrepresentable problems are called hard problems. Such problems arise if the function φ relating the output and input (b, $= \varphi(\mathbf{a}_l)$) is not linearly separable. In

(a)

Input		Output
a1	a2	b1
0	0	0
0	0	0
1	0	0
1	1	1



(b)

Input		Output
a1	a2	b1
0	0	1
0	1	0
1	0	0
1	1	1

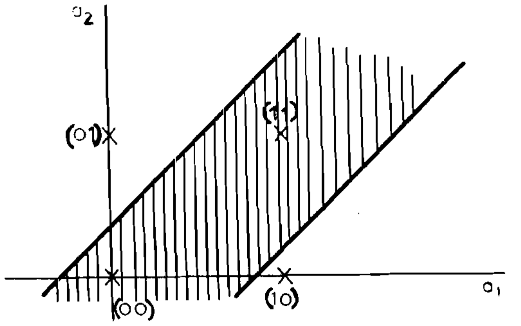


Figure 9. Two 2-class problems to illustrate linear separability–linearly separable **(a)** and unseparable **(b)** cases.

geometrical terms linear separability means that the given set of input patterns **{a,)** can be separated into M distinct regions in the N-dimensional pattern space by a set of linear hyperplanes. Here M corresponds to the number of distinct output patterns or classes. As a simple illustration, we can consider **2-dimensional** binary **(0, 1)** patterns in input pattern space and a 1-dimensional output pattern. Two pattern classification problems are shown in figure 9. Note that the number of input patterns (4) is more than the number of dimensions (2) of the input pattern space. These are two-class problems, as the number of distinct outputs are two. Of the two problems in the figure, the first one is linearly separable since a straight line separates the patterns into two regions of desired classes. In the second problem the desired region cannot be obtained by using a single straight line. Note that a straight line is equivalent to a linear hyperplane in a 2-dimensional space.

The restriction of linear separability is due to the function relating input and output patterns. Any arbitrary assignment of an output pattern to a set of input patterns need not result in a linearly separable function, and hence cannot be represented by the two layer network with nonlinear units in the output layer. Thus, although the restriction on the number and type of input patterns (as in the case of pattern association problem) is removed due to introduction of nonlinear units, a restriction is now placed on the nature of the function relating the input and output patterns. To remove this restriction a multilayer feedforward network with nonlinear processing units can be used (**Minsky** & **Papert** 1988). Such a network can handle a more general class of pattern classification problems, namely, ***pattern mapping*** problems which will be discussed in the next section. Geometrically, it can be argued that a multilayer feedforward neural network can perform classification of patterns with complex boundary surfaces separating different classes in an N-dimensional space (Lippmann 1987; Minsky & Papert 1988). However, training such a network is not straightforward. Thus it leaves us with a ***hard learning problem*** which can solved using the ***generalized delta rule*** (Rumelhart & McClelland 1986).

4.1c ***Pattern mapping:*** For a pattern mapping problem the input and output patterns are points in the N- ***and*** M-dimensional continuous spaces, respectively. The objective is to capture the implied functional relationship or mapping function between the input and output by training a feedforward neural network. This is also called the ***generalization*** problem (Deuker ***et al*** 1987). Once the network generalizes by capturing the mapping function through its weights, then during recall from an input pattern the network produces an output which is an interpolated version of the outputs of the training input patterns near the current input pattern. The input patterns are generally naturally occurring patterns as in speech and hand-printed characters.

A multilayer feedforward network with at least two intermediate layers in addition to the input and output layers can perform a pattern mapping task (Cybenko 1989). The number of units in the input and output layers correspond to the dimensions of the input and output patterns, respectively. The additional layers are called ***hidden layers,*** and the number of units in a hidden layer is determined depending on the problem, usually by trial and error. The network can be trained (i.e. weights at different layers can be adjusted) for a given set of input–output pattern pairs using a ***generalized delta rule*** or ***backpropagation law*** (see figure 10) (Rumelhart & McClelland 1986; Hush & Horne 1993). It is derived using the principle of gradient descent along the error surface in the weight space. The given patterns are applied in some random order one by one, and the weights are adjusted using the backpropagation law. The pattern pairs may have to be applied several times till the output error is reduced to an acceptable value.

Once the network is trained, it can be used to recall the appropriate pattern (in this case some interpolated output pattern) for a new input pattern. The computation is straightforward in the sense that the weights and the output functions of the units at different layers are used to compute the activation values and output signals. The signals from the output layer correspond to the output.

Note that for the backpropagation law to work (see figure 10), the output function of the units in the hidden and output layers must be nonlinear and differentiable. Such functions are called ***semilinear.*** If they are linear, no advantage is obtained by using additional hidden layers. By using a hardlimiting threshold function, it is not possible to propagate the error to hidden layer units to adjust the weights in that layer. Thus the advantage of complex pattern mapping or pattern classification is obtained by a multilayer feedforward network mainly because of the use of the semilinear output functions.

The use of semilinear functions results in a rough error surface in the weight space. That is, there will be several local minima, besides a global minimum. The effects of local minima can be partially reduced by using a stochastic update of weight values (Wasserman 1988). In general the backpropagation learning law needs several iterations in order to reach an acceptably low value of error, at which the network can be assumed to have captured the implied mapping in the given set of input–output

Backpropagation algorithm: Generalized delta rule

Given a set of input–output patterns $a,, b,,$ I $= 1, 2, \ldots L$
Ith input vector $a, = (a,,, a,,, \ldots a_{lN})^T$ and output vector $\mathbf{b}_l = (b_{l1}, b_{l2}, \ldots b_{lN})^T$
Assume only one hidden layer and initial setting of weights to be arbitrary
Assume input layer with only linear units. Then output signal = input activation value
η is the learning rate parameter
Activation of unit i in the input layer $x_{li} = a_{li}$
Activation of unit j in the hidden layer $x^h_{lj} = \sum_{i=l}^{N} w^h_{ji} x_{li} + \theta^h_j$
Output signal from the jth unit in the hidden layer, $s^h_{lj} = f^h_j(x^h_{lj})$
Activation of unit k in the output layer $x^0_{lk} = \sum_{j=1}^{H} w^0_{kj} s^h_{lj} + \theta^0_k$
Output signal from unit k in the output layer $s^0_{lk} = f^0_k(x^0_{lk})$
Error term for the kth output unit $\delta^0_{lk} = (b,, - s^0_{lk}) f^{0'}_k(x^0_{lk})$
Update the weights on the output layer $w^0_{kj}(t + 1) = w^0_{kj}(t) + \eta \delta^0_{lk} s^h_{lj}$
Error term for the jth hidden unit $\delta^h_{lj} = f^{h'}_j(x^h_{lj}) \sum_{k=1}^{M} \delta^0_{lk} w^0_{kj}$
Update the weights on the hidden layer $w^h_{ji}(t + 1) = w^h_{ji}(t) + \eta \delta^h_{lj} a_{li}$
Calculate the error for the Ith pattern $E_l = \frac{1}{2} \sum_{k=1}^{M} \delta^2_{lk}$
Total error for all patterns $E = \sum_{l=1}^{L} E_l$
Apply the given patterns, may be several times, in some random order and update the weights until the total error reduces to an acceptable value.

Figure 10. Generalized delta rule.

pattern pairs. However, due to the slow rate of convergence of the backpropagation learning law, new architectures (like counter propagation, Hecht-Nielsen 1990) are being sought for faster learning.

4.2 *Pattern recognition tasks by feedback* ANN

4.2a *Autoassociation:* In this section we consider pattern recognition tasks that can be performed by simple feedback neural networks (figure 11). We begin with the autoassociation task discussed earlier when the input and output patterns in each pair are the same i.e., a, = b,, $1 = 1, 2 \ldots$ L. The objective in an autoassociation task is to design a network that can recall a stored pattern given a corrupted (noisy or partial) version of the pattern. A feedback network with N linear processing units can perform the task of autoassociation. Such a network can be trained (i.e., the weights can be determined) using either Hebb's law or delta rule (Hecht-Nielsen 1990). Hebb's learning law leads to a set of optimal weights when the given patterns are orthogonal. Delta rule leads to set of optimal weights when the given patterns are linearly independent.

Pattern recall will be exact when the test pattern is same as one of the stored ones, represented by the weights. If the test pattern is a noisy version of the stored pattern, the recalled pattern is also a noisy version of the stored pattern. In fact the network recalls the input pattern itself, as every vector is associated with itself, thus completely eliminating any accretive behaviour (Murakami & Aibara 1987).

Auto association (Pattern storage)

- *Arch:* Single layer with feedback, linear processing units
- *Learning:* Hebb (orthogonal inputs), Delta (linearly independent inputs)
- *Recall:* Direct
- *Limitation:* Linear independence of patterns, # of patterns limited to dimensionality
- *To Overcome:* Nonlinear processing units, becomes a pattern storage problem

Pattern storage

- *Arch:* FBNN, nonlinear processing units, states, Hopfield energy analysis
- *Learning:* Not important
- *Recall:* Activation dynamics until stable states are reached
- *Limitation:* False minima, hard problems, limited # patterns
- *To Overcome:* Stochastic update, hidden units.

Pattern environment storage

- *Arch:* Boltzmann machine, nonlinear processing units, hidden units, stochastic update
- *Learning:* BM learning law, simulated annealing
- *Recall:* Activation dynamics, simulated annealing
- *Limitation:* Slow learning
- *To Overcome:* Different architecture

Figure 11. Pattern recognition tasks by feedback ANN (FBNN).

Thus autoassociation by a feedback network with linear units is not going to serve any purpose. Moreover, the number of patterns is limited to the dimensionality of the pattern. Although there is no simple learning law, it can be shown that the weights of such a network can be determined to store any $L \leqslant N$ patterns, without any error in recall, where N is the dimension of the input pattern space. Discussion of autoassociation task by a feedback network with linear units is only of academic interest, as any input pattern comes out as itself if it is one of the stored ones, and a noise input comes out as a noisy pattern, not as the nearest stored pattern.

To overcome this limitation due to the absence of accretive behaviour, the linear units are replaced with units having nonlinear output functions. The resulting feedback network can then perform ***pattern storage*** task which will be considered next.

4.2b ***Pattern storage:*** The objective is to store a given set of patterns so that any one of the patterns can be recalled exactly when an approximate (corrupted) version of the pattern is presented to the network. What is needed is the storage of features and their spatial relations in the patterns, and the pattern recall should take place even when the features and their spatial relations are slightly modified due to noise and distortion. The approximation of pattern refers to the closeness of the features and their spatial relations in the pattern when compared to the original stored pattern. What is actually stored in practice is the information in the pattern data itself. The approximation is measured in terms of some distance, like Hamming distance (in case of binary patterns). The distance feature is automatically realized through the threshold (binary) feature of the output function of a processing unit. The pattern storage is accomplished by a feedback network consisting of nonlinear processing units (see figure 12).

For the simplest case, the weights on the connecting links between units are assumed to be symmetric, i.e., $w_{tf} = w_{ji}$, and that there is no self feedback, i.e., $w_{ii} = 0$. The output signals of all units at any instant of time define the state of the network at that instant. Each state of the network can be assumed to correspond to some ***energy***

Hopfield net algorithm – To store and recall a set of bipolar patterns
Let the network consist of N fully connected units with each unit having hard limiting bipolar threshold output function.

Let {a,), 1 = 1, 2 . . . L be the vectors to be stored.

The vectors {a,) are assumed to have bipolar components, i.e., $a_{li} = \pm 1$.

1. Assign the connection weights

$$w_{ij} = \sum_{l=1}^{L} a_{li}a_{lj}, \quad \text{for} \quad i \neq j$$

$$= 0, \quad \text{for} \quad i = j, \quad 1 \leqslant i,j \leqslant L.$$

2. Initialize the network output with the given unknown input pattern a

$$s_i(0) = a_i, \quad i = 1, 2 \ldots N$$

where $s_i(0)$ is the output of the unit i at time $t = 0$.

3. Iterate until convergence

$$s_i(t+1) = sign\left[\sum_{j=1}^{N} w_{ij}s_j(t)\right], \quad i = 1, 2 \ldots N$$

The process is repeated until the outputs remain unchanged with further iteration. The steady outputs of the units represent the stored pattern that best matches the given input.

Figure 12. Hopfield Net algorithm to store and recall a set of bipolar patterns.

which is defined in terms of the output state $\{s_i\}$ and weights $[w_{ij}]$ of the network as (Hopfield 1982)

$$E = -\tfrac{1}{2}\sum_{i}\sum_{j \neq i} s_i w_{ij} s_j - \sum_{i} I_i s_i + \sum \theta_i s_i,$$

where I_i is an external input and θ_i is the threshold of the unit. The energy as a function of the output state can be viewed as something like an ***energy landscape.*** The shape of the landscape is dictated by the network units and their interconnection strengths (weights). The feedback and the nonlinear processing units of the network create ***basins of attraction*** in the energy landscape. The basins tend to be regions of equilibrium states. If there is a fixed state (point in the output state space) in each of these basins where the energy is minimum, these states corresponds to fixed points of equilibrium. There could also be periodic (or oscillating) regions or chaotic regions of equilibrium (Kosko 1992).

It is the existence of the basins of attraction that is exploited to store the desired patterns and recall them even with approximate inputs as keys. Each pattern is stored at a fixed point of equilibrium of the ***energy minimum.*** An erroneous or distorted pattern is more likely to be closer to the corresponding true pattern than to the other stored patterns. Each input pattern results in a state of the network that may be closer to the desired state, in the sense that it may lie near the basin of attraction corresponding to the true state. Since an arbitrary state need not correspond to a stable state, the activation dynamics of the network may eventually lead to a ***stable state*** from which the desired pattern may be read or derived.

If the nonlinear output function of each unit is a binary threshold function (curve A

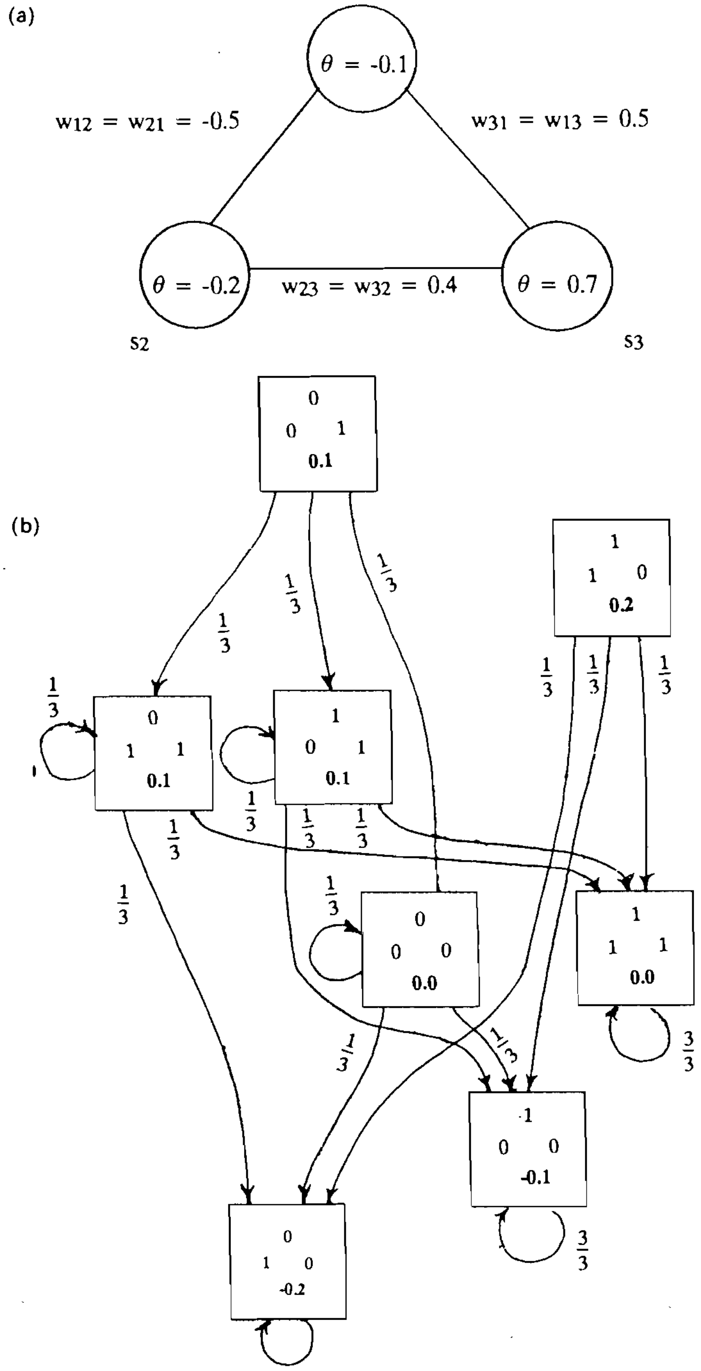


Figure 13. **(a)** A 3-unit feedback network with symmetric weights and binary threshold units. Activation dynamics $x_j = \sum_j w_{ji}s_i - \theta_j$, $s_j = f(x_j)$; Energy $E = -\frac{1}{2}\sum_i \sum_i w_{ij}s_i s_j + \sum s_i \theta_i$. **(b)** State transition diagram for the 3-unit network of figure 13a. Each block represents a state given by sequence s_1, s_2, s_3. There are eight blocks for eight states. The energy for each state is indicated by the bold numbers with each block. Note that the state diagram has three stable states *(**111, 100** and **010**)* (Aleksander & Morton ***1990)**.*

in figure 3), then the stable stables of the network would lie at the corners of the binary hypercube in the N-dimensional discrete binary space. On the other hand, if the output function is a semilinear function (curve C in figure 3), then the points corresponding to these states may move closer to each other within the unit hypercube. If the output function is a horizontal line, then almost all states remain close to each other, and hence there will be only one state for the network.

Given a network, it is possible to determine the state transition diagram (Aleksander & Morton 1990). Figure 13 shows the state transition diagram for a 3-unit network. The diagram illustrates the different states of the network and their transition probabilities. States which have self transition with probability 1 are stable states. For a given number of units, the state transition probabilities and the number or stable states are dictated by the connection strengths or weights.

Since each state is associated with some energy value, the state transition diagram shows transitions from a state with higher energy value to a state having lower or equal energy value. The energy value of a stable state corresponds to an energy minimum in the landscape, as there is no transition from this to the other states.

The number of basins of attraction in the energy landscape depends only on the network, **i.e.**, the number of processing units and their interconnection strengths (weights). When the number of patterns to be stored is less than the number of basins of attraction, **i.e.**, stable states, then there will be spurious stable states, which do not correspond to any desired patterns. In such a case, when the network is presented with an approximate pattern for recall, the activation dynamics may eventually lead to a stable state which may correspond to one of the spurious states or a ***false energy minimum,*** or to one of the stable states corresponding to some other pattern. In the latter case there will be an undetected error in the recall. The average probability of error depends on the energy values of the stable states corresponding to the desired patterns, and the relative locations of these states in the state space, measured in terms of some distance criterion.

If the number of desired patterns to be stored is more than the number of basins of attraction in the energy landscape, then the problem becomes a ***hard problem,*** in the sense that the given patterns cannot be stored in the network.

For a given network it is not normally possible to determine exactly the number of basins of attraction as well as their relative spacings and depths in the state space of the network. It is possible to estimate the ***capacity*** (number of patterns that can be stored) of the network and also the average ***probability of error*** in recall (Abu-Mostafa & St. Jaques 1985; Aleksander & Morton 1990). The probability of error in recall can be reduced by adjusting the weights in such a way that the resulting energy landscape is matched to the probability distribution of the input patterns. This becomes the problem of storing a ***pattern environment.***

4.2c ***Pattern environment storage:*** A pattern environment is described by the set of desired patterns together with their probability distribution. The objective is to store a pattern environment in a network in such a way that the average probability of error in recall is minimized. This is achieved if the energy landscape is designed in such a way that the desired patterns are stored at the stable states corresponding to the lowest minima, with the higher probability patterns at lower energy minima points.

Boltzmann machine architecture together with the Boltzmann learning law can achieve an optimal storage of pattern environment **(Hinton** & Sejnowski 1986;

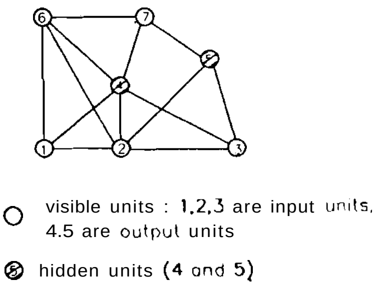


Figure 14. Architecture of the Boltzmann machine. Each unit is connected to every other unit, although only a few connections are shown in the figure. Some of the visible units can be identified as input units and others as output units if the machine is to be used for pattern mapping.

Aleksander & Morton 1990). The architecture consists of a number of processing units with each unit connecting to all the other units (figure 14). The number of units is typically larger than the dimension of the input pattern. The additional units are called ***hidden units.*** Use of hidden units helps in overcoming the limitation of the hard problems of pattern storage by a fully connected network. The patterns are applied to the so-called ***visible*** units, the number of visible units being equal to the dimension of the input patterns.

Error in pattern recall due to false minima can be reduced significantly if initially the desired patterns are stored (by careful training) at the lowest energy minima. The remaining error can be reduced by using suitable activation dynamics. Let us assume that by training we have achieved a set of weights which will enable the desired patterns to be stored at the lowest energy minima. The activation dynamics is modified so that the network can also move to a state of higher energy value initially, and then to the nearest deep energy minimum. It is possible to realize this by using a ***stochastic update*** in each unit instead of the deterministic update of the output function as in the previous cases. By stochastic update we mean that the activation value or the net input to a unit need not decide the next output state of the unit in a deterministic manner as in the case of figure 12. The update is expressed in probabilistic terms, like the probability of firing the unit being greater than 0.5 if the net input exceeds a threshold, and less than 0·5 if the net input is less than the threshold for the unit. Note that the output function could still be a threshold logic (hardlimiter), but it is applied in a stochastic manner.

With the new activation dynamics, the state transition diagram shows transitions from a lower energy state to a higher energy state as well, the probability of such a transition is dictated by the probability function used in determining the firing of a unit in the stochastic update (Aleksander & Morton 1990). The probability function (figure 15) can in turn be defined in terms of a parameter, called ***temperature*** (T). As the temperature is increased, the uncertainty in the update increases, giving the network a greater chance to go to a higher energy level state.

Since eventually we want the activation dynamics to lead the network to a stable state corresponding to the pattern closest to the given input pattern, we need to provide greater mobility for transition to higher states only initially. The mobility is slowly decreased by reducing the temperature, eventually to $T = 0$. At the lowest temperature the network settles down to a fixed point state corresponding to the desired pattern. At each temperature the network dynamics is allowed to settle to some equilibrium situation, called ***thermal equilibrium.*** At thermal equilibrium the average probability of visiting the states of the networks will not change further. The temperature parameter is reduced in a predetermined manner (called annealing

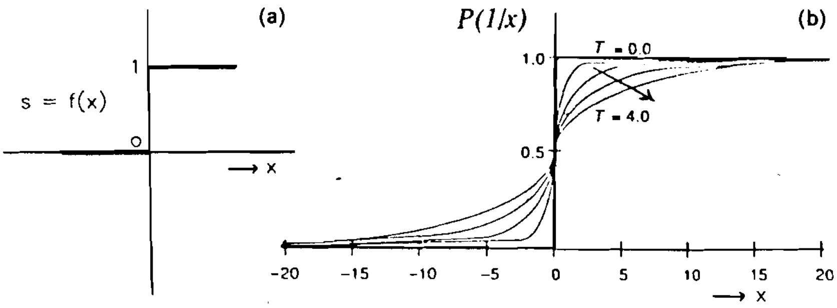


Figure 15. Stochastic update of a unit using probability law. Probability of firing $P(1/x) = 1/[1 + \exp(-x/T)]$. (a) Binary output function; (b) probability function for stochastic update.

schedule), making sure that at each temperature the network is allowed to reach thermal equilibrium before the next change in temperature is made. This process is called ***simulated annealing*** (Kirkpatrick ***et al*** 1983) (see figure 16). Note that at each temperature the state update dynamics is fixed, as the probability of transition from one state to another depends only on the temperature. The update dynamics is however altered when the temperature is changed, and it results in a new state transition diagram.

The states at thermal equilibrium at T = 0 represent the stable states of the network corresponding to the minima of the energy landscape. The probabilities of these states are also related to the actual minimum energy values of the states. The relation between the probabilities of stable states and energy suggest that the probability of error in the recall of patterns can be further reduced if the probability distribution of the desired patterns, i.e., the pattern environment, is known, and is used in determining the optimal setting of weights of the network.

Simulated annealing algorithm – To recall a stored pattern with partial input

Let us assume a **Boltzmann** machine with some visible units and some hidden units.

Let the network consist of total N fully connected units, with each unit having a hard limiting binary threshold output function. Let us assume that the network was already trained to store the given set of input patterns.

1. Force the outputs of the visible units to the corresponding known **components** in the given partial binary input vector.
2. Assign for all unknown visible units and all hidden units to random binary output values.
3. Select a unit k at random, and calculate its activation value x_k using weighted sum of its inputs.
4. Assign the output of the unit k to 1 with probability $P_k = \frac{1}{1 + \exp(-x_k/T)}$, where T is the temperature parameter.
5. Repeat steps 3 and 4 until all units have had the same probability of being selected for update. This number of unit-updates defines a processing cycle.
6. Repeat step 5 for several processing cycles until ***thermal equilibrium*** has been reached at the given temperature *T*, i.e, when the probability of visiting different states of the network does not change any further. This is usually accomplished only approximately.
7. Lower the temperature, and repeat steps 3 through 7 until a stable state is reached at which point there will not be any further change in the state of the network. The result of recall is the stable output state of the visible units.

Figure 16. Simulated annealing algorithm.

Learning in Boltzmann machine

The objective is to adjust the weights of a Boltzmann machine so as to store a pattern environment described by the set of vectors $\{V_a\}$ and their probabilities of occurrence. These vectors should appear as the outputs of the visible units. Define $\{H_b\}$ as the set of vectors appearing on the hidden units.

Let $P^+(V_a)$ be the probability that the outputs of the visible units will be clamped (indicated by "+" superscript) to the vector V_a. Then,

$$P^+(V_a) = \sum_b \mathrm{P}^+(V_a \wedge H_b),$$

where $P^+(V_a \wedge H_b)$ is the probability of the state of the network when the outputs of the visible units are clamped to the vector V_a, and the outputs of the hidden units are H_b.

Likewise the probability that V_a will appear on the visible units when none of the visible units are clamped (indicated by "−" superscript) is given by

$$P^-(V_a) = \sum_b P^-(V_a \wedge H_b).$$

Note that $P^+(V_a)$ is given by the pattern environment description, and $P^-(V_a)$ depends on the network dynamics and is given by

$$P^-(V_a) = \sum_b \exp(-E_{ab}/T) \Big/ \sum_{m,n} \exp(-E_{mn}/T),$$

where the total energy of the system in the state $V_a \wedge H_b$ is given by

$$E_{ab} = -\tfrac{1}{2} \sum_{i,j} w_{ij} s_i^{ab} s_j^{ab},$$

s_i^{ab} refers to the output of the ith unit in the state $V_a \wedge H_b$.

The Boltzmann learning law is derived using the negative gradient descent of the functional

$$G = \sum_a P^+(V_a) \ln[P^+(V_a)/P^-(V_a)].$$

It can be shown that

$$-\partial G/\partial w_{ij} = (1/T)(P_{ij}^+ - P_{ij}^-),$$

where

$$P_{ij}^+ = \sum_{a,b} P^+(V_a \wedge H_b) s_i^{ab} s_j^{ab},$$

$$P_{ij}^- = \sum_{a,b} P^-(V_a \wedge H_b) s_i^{ab} s_j^{ab}.$$

The weight updates are calculated according to

$$\Delta w_{ij} = -\eta(\partial G/\partial w_{ij}) = \eta(1/T)(P_{ij}^+ - P_{ij}^-).$$

The Boltzmann law is implemented using some annealing schedule for the network during clamped and unclamped phases of the visible units of the network to determine P_{ij}^+ and P_{ij}^-, respectively.

Figure 17. Boltzmann learning law.

The Boltzmann learning law (see figure 17) allows us to represent a given environment by the network (Ackley *et al* 1985; Hinton & Sejnowski 1986; Aleksander & Morton 1990). The law uses an information theoretic measure to evaluate how well the environment is represented in the network. If a perfect representation is obtained, then there will be as many energy minima as there are desired patterns. But in practice only an approximate representation of the environment is accomplished, and hence there will be some spurious stable states which correspocd to the false wells in the energy landscape. The Boltzmann learning law uses a simulated annealing schedule for implementation, i.e., for determining the weight updates at each stage. Recall of stored patterns from an approximate input pattern also uses a simulated annealing schedule to overcome the false minima created because of the approximate representation of the environment by the network.

In general the Boltzmann learning law converges slowly to the desired weights (Geman & Geman 1984; Szu 1986, pp. 420–5). Moreover, there is no simple way to determine the optimum number of hidden units for a network to solve the given problem of pattern environment storage. The larger the number of hidden units, the greater is the chance for more false minima, and hence the greater the probability of error in recalling a stored pattern. The smaller the number of hidden units, the greater the chance that the given problem becomes hard for the network. New architectures are needed to overcome some of these limitations of the Boltzmann machine for the problem of pattern environment storage.

4.3 *Pattern recognition tasks by feedforward and feedback* ANN

In this section we discuss some pattern recognition tasks (figure 18) that can be performed by a network consisting of two layers of processing units: The first layer with linear output units feeds the input pattern to the units in the second layer through a set of feedforward connections with appropriate weights. The outputs of

Pattern storage (STM)

* *Arch:* **Two layers (input & competitive), linear processing units**
* *Learning:* **No learning in FF stage, fixed weights in FB layer**
* *Recall:* **Not relevant**
* *Limitation:* **STM, no application, theoretical interest**
* *To overcome:* **Nonlinear output function, learning in FF stage**

Pattern clustering (grouping)

* *Arch:* **Two layers (input & competitive), nonlinear processing units**
* *Learning:* **Only in FF stage – Competitive learning**
* *Recall:* **Direct, activation dynamics until stable state is reached**
* *Limitation:* **Fixed (rigid) grouping of patterns**
* *To overcome:* **Neighbourhood units in competition layer**

Feature map

* *Arch:* **Self-organization network, 2 layers, nonlinear processing units**
* *Learning:* **Neighbourhood units in competitive layer**
* *Recall:* **Apply input, determine winner**
* *Limitation:* **Only visual features, not quantitative**
* *To overcome:* **More complex architecture**

Figure 18. Pattern recognition tasks by feedforward (FF) and feedback (FB) ANN.

the units in the second layer are fedback to the units in the same layer including feedback to the same unit. The self feedback is usually with a positive weight (excitatory connection) and the feedback to the other units is usually with a negative weight (inhibitory connection). The weights on the feedback connections in the seond layer are usually fixed. The first layer of units is called input layer, and the second layer is called competitive layer (Rumelhart & Zipser 1986). Different choices of output functions and methods of learning lead to networks for different types of competition tasks. We discuss three such tasks. Assuming fixed weights in the feedforward connections from the input to the competitive layer, and in the feedback connections in the competitive layer, we can study the behaviour of the network for different types of output functions of the units in the competition layer.

4.3a Pattern storage (short term memory): First let us assume the output functions to be linear. When an input pattern is applied, the units in the competition layer settle to a steady activation state which will remain there even after the input pattern is removed (Freeman & Skupura 1991). The activation pattern will remain as long as the network is not given a different input pattern. Another input pattern will erase the previous activation state. Hence this is called short-term memory. The pattern is stored only temporarily.

This pattern storage representation is only of theoretical interest. There is no application for such a short-term memory function. However, by using a nonlinear output function for the units in the competition layer one could show that the network can perform a pattern clustering task.

4.3b Pattern clustering: Given a set of patterns, the objective is to design a competition network which groups the patterns into subgroups of patterns based on similarity of features in the patterns. A two layer network with input and competition layers, and with nonlinear units in the competition layer can perform the task of pattern clustering or grouping (Grossberg 1980).

If a nonlinear output function of the type $f(x) = x^2$ is used for the units in the competitive layer, then it can be shown (Freeman & Skupura 1991) that the activation dynamics leads to a steady state situation where the network tends to enhance the activity of the unit with the largest activity. When the input pattern is removed, the activities of all units except the largest one will decay to zero. Thus only one of the units in the competitive layer will win. The weights leading to the winning unit j are adjusted to respond more to the input pattern a. This weight adjustment is repeated for all the input patterns several times. For input patterns belonging to different groups, different units in the competition layer will win. When the weight vector for each output unit reaches an average position within the cluster, it will stay generally within a small region around that average position. Each unit in the competitive layer refers to a different group or category of patterns.

When an unknown input pattern is given, the activation dynamics leads to a steady state situation where only one unit in the competitive layer is active. That unit gives the category to which the input pattern belongs.

Note that in a competitive network the physical location of the units do not reflect any relation between categories. But there are many situations where the patterns do not fall into fixed categories. There may be a gradual change of features from one pattern to another. This change of features can be captured by a selforganisation network which performs the task of feature mapping (von der Malsburg 1973; Willshaw & von der Malsburg 1976).

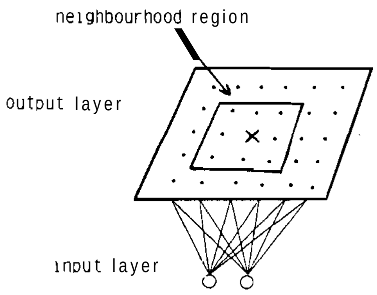


Figure 19. A feature mapping architecture. Each input unit is connected to all the units of the output layer.

4.4b ***Feature map:*** Given a set of patterns, the objective is to design a network that would organise the patterns in accordance with similarity of features among them in such a way that by looking at the output of the network one can visually obtain an idea of how different patterns are related. The display of signals from the output layer (typically in 2-dimension) of units is called a ***feature map.***

To accomplish the task of feature mapping, a competitive network is modified into one called a ***selforganising network*** (Kohonen 1990; Freeman & Skupura 1991) shown in figure 19. The modification consists of creating a neighbourhood region around the winning unit in the competitive layer, so that during training all the feedforward weights leading to the units in this region are adjusted to favour the input pattern. The weight adjustment is similar to the case of a competitive network. The neighbourhood region around a winning unit is gradually reduced for each application of the given set of patterns (see figure 20).

Algorithm for self-organizing feature map

1. Initialize the weights from N inputs to the M output units to small random values. Initialize the size of the neighbourhood region $\boldsymbol{R}(0)$.

2. Present a new input **a**

3. Compute the distance d_i between input and the weight on each output unit i:

$$d_i = \sum_{j=1} [a_j(t) - w_{ij}(t)]^2, \quad \text{for} \quad i = 1, 2 \ldots M$$

where $a_j(t)$ is the input to the jth input unit at time t and $w_{ij}(t)$ is the weight from the jth input unit to the ith output unit.

4. Select the output unit i* with minimum distance

$$i^* = \text{index of} \left[\min_i (d_i)\right]$$

5. Update weight to node i* and its neighbours

$$w_{ij}(t+1) = w_{ij}(t) + \eta(t)(a_i(t) - w_{ij}(t)),$$
$$\text{for } i \in R_i^*(t), \text{ and } i = 1, 2 \ldots N,$$

where $\eta(t)$ is the learning rate parameter $(0 < \eta(t) < 1)$ that decreases with time $R_i^*(t)$ gives the neighbourhood region around the node i*, at time t.

6. Repeat steps 2 through 5 for all inputs several times.

Figure 20. An algorithm for self-organizing feature map.

For recall, when an unknown input is applied, the activation dynamics determines the winning unit whose location would determine its features relative to the features represented by the other units in its neighbourhood.

While a feature map produces a more realistic arrangement of patterns, the output is useful only for visual observation. Since it is difficult to categorize a feature map, it is difficult to use it for applications such as pattern classification. A more complex architecture is needed to exploit the advantages of a feature map for pattern classification purposes (Huang & Kuh 1992).

5. Architectures for complex pattern recognition tasks

So far we have considered simple structures of neural networks and discussed the pattern recognition tasks that these structures could accomplish. In practice the pattern recognition tasks are much more complex, and each task may require evolving a new structure based on the principles discussed in the previous sections. In fact designing an architecture for a given task involves developing a suitable structure of the neural network and defining appropriate activation and synaptic dynamics. In this section we will discuss some general architectures for complex pattern recognition tasks.

5.1 *Associative memory: pattern storage – BAM*

Pattern storage is the most obvious pattern recognition task that one would like to accomplish by an ANN. This is a memory function, where the network is expected to store the pattern information for later recall. The patterns to be stored may be spatial or spatiotemporal (pattern sequence). Typically an ANN behaves like an associative memory, in which a pattern is associated with another, or with itself. This is in contrast with the random access memory which maps an address to a data. An ANN can also function as a content addressable memory where data are mapped to an address.

The pattern information is stored in the weight matrix of a feedback neural network. The stable states of the network represent the stored patterns, which can be recalled by providing an external simulus in the form of partial input. If the weight matrix stores the given patterns, then network becomes an autoassociative memory. Several architectures are proposed in the literature for realizing an associative memory function depending on whether the pattern data is discrete/continuous, or the network is operating in discrete time/continuous time, or the learning is taking place off-line/on-line (Simpson 1990).

We will discuss the discrete bidirectional assoċiative memory (BAM) in some detail. It is a two-layer heteroassociative neural network (figure 21) that encodes arbitrary binary spatial patterns using Hebbian learning. It learns on-line and operates in discrete time. The BAM weight matrix is given by,

$$W = \sum_{l=1}^{L} \mathbf{a}_l{}^T \mathbf{b}_l$$

where $\mathbf{a}_l \in \{-1, +1\}^N$ and $\mathbf{b}_l \in \{-1, +1\}^N$. The superscript T refers to the transpose of the vector.

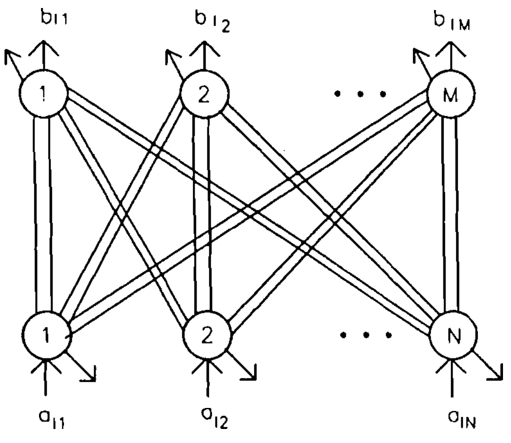


Figure 21. Discrete bidirectional associative memory.

The activation equations are as follows:

$$b_j(t+1) = \begin{cases} 1, & \text{if } y_j > 0, \\ b_j(t), & \text{if } y_j = 0, \\ -1, & \text{if } y_j < 0, \end{cases} \qquad \text{where} \quad y_j = \sum_{i=1}^{N} a_i(t) w_{ji},$$

$$a_i(t+1) = \begin{cases} 1, & \text{if } x_i > 0, \\ a_i(t), & \text{if } x_i = 0, \\ -1, & \text{if } x_i < 0, \end{cases} \qquad \text{where} \quad x_i = \sum_{j=1}^{N} b_j(t) w_{ij}.$$

For recall, the given input $a_i(0)$, $i = 1, 2 \ldots N$ is applied and the stable values of $b_j(\infty)$, $j = 1, 2 \ldots M$ are 'read out. BAM updates are synchronous in the sense that the units in each layer are updated simultaneously.

BAM can be shown to be unconditionally stable (Kosko 1988). However its storage is limited to a small number of **binary/bipolar** patterns.

5.2 Pattern mapping: Data compression – *CPN*

In pattern mapping the objective is to capture the implied functional relationship between an input–output vector pair (**a**,, b,). We have seen earlier that a multilayer feedforward network with a semilinear output function can perform generalization, but the training process is slow, and the ability to generalize depends on the learning rate and the number of units in the hidden layers. Several architectures are proposed in literature for realizing a mapping function (**Simpson** 1990). A practical approach for implementing pattern mapping is to use an architecture that learns fast. A **counter-propagation** network (CPN) that uses a combination of **instar** and **outstar** topologies is proposed (figure 22) for this purpose (Hecht-Nielson 1987). It consists of a **three-layer** feedforward network with the first two layers forming a competitive learning system and the second (hidden) and third layers forming an **outstar** structure. Learning takes place in the **instar** structure of the competitive learning system to code the input patterns **{a,)** and in the **outstar** structure to represent the output patterns {b,). The training of the **instar** and **outstar** structures are as follows.

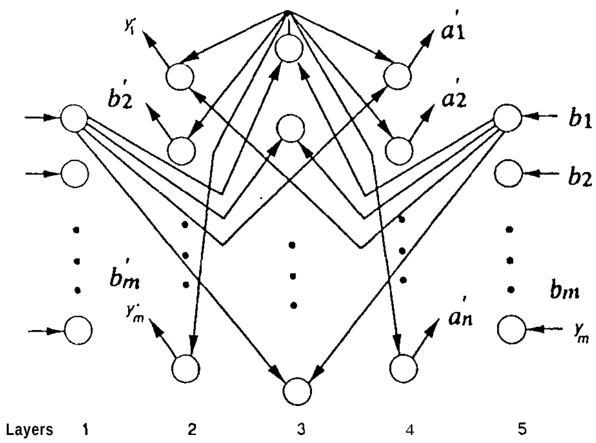


Figure 22. Counter propagation network.

Training instars of CPN

(1) Select an input vector $\mathbf{a}_l$ from the given training set $(\mathbf{a}_l, \mathbf{b}_l)$, $l = 1, 2, \ldots \mathbf{N}$.
(2) Normalize the input vector and apply it to the CPN competitive layer.
(3) Determine the unit that wins the competition by determining the unit m whose vector w is closest to the given input.
(4) Update the winning unit's weight vector as

$$\mathbf{w}_m(t+1) = \mathbf{w}_m(t) + \eta(\mathbf{a}_l - \mathbf{w}_m(t)).$$

(5) Repeat steps 1 through **4** until all input vectors are grouped properly by applying the training set several times.

After successful training the weight vectors leading to each hidden unit represent the average of the input vectors corresponding to the group represented by the unit.

Training outstars of CPN

(1) After training instars, apply a normalized input vector $\mathbf{a}_l$ to the input layer and the corresponding output $\mathbf{b}_l$ to the output layer.
(2) Determine the winning unit m in the competitive layer.
(3) Update the weights on the connections from the winning competitive unit to the output units

$$\mathbf{w}_m(t+1) = \mathbf{w}_m(t) + \beta(\mathbf{b}_l - \mathbf{w}_m(t)).$$

(4) Repeat steps 1 through 3 until all the vector pairs in the training set are mapped satisfactory.

After successful training, the outstar weights for each unit in the competitive layer represents the average of the subset of the output vectors $\mathbf{b}_l$ corresponding to the input vectors belonging to that unit.

Depending on the number of nodes in the hidden layer, the network can perform any desired mapping function. In the extreme case, if a unit is provided in the hidden layer for each input pattern, then any arbitrary mapping **(a,,b,)** can be realized. But in such a case the network fails to generalize. It merely stores the pattern pair. By using a small number of units in the hidden layer, the network can accomplish data compression. Note also that the network can be trained to capture the inverse mapping as well, i.e., **a**, = $\phi^{-1}(\mathbf{b}_l)$, provided such a mapping exists and it is unique. The name counterpropagation is given due to the network's ability to learn both forward and inverse mapping functions.

5.3 *Pattern classification: stability-plasticity dilemma* – *ART*

Many pattern mapping networks can be transformed to perform pattern classification or category learning tasks. However these networks have the disadvantage that during learning the weight vectors tend to encode the presently active pattern, thus weakening the traces of patterns it had already learnt. Moreover any new pattern that does not belong to the categories already learnt, is still forced into one of them, using the best match strategy without taking into account how good even the best match is. The lack of stability of weights as well as lack of inability to accommodate patterns belonging to new categories, led to the proposal of new architectures for pattern classification. These architectures are based on adaptive resonance theory (ART) and are specially designed to take care of the so called ***stability–plasticity dilemma*** in pattern classification (Carpenter & Grossberg 1988).

ART also uses a combination of instar–outstar network as in CPN, but with the output layer merged with the input layer, thus forming a two-layer network with feedback as shown in figure 23. The minimal ART network includes a bottom-up competitive learning system (F_1 to F_2) combined with a top-down (F_2 to F_1) outstar pattern learning system. The number of units in the F_2 layer determines the number of possible categories of input patterns. When an input pattern a, is presented to the F_1 layer, the system dynamics initially follows the course of competitive learning, leading to a winning unit in the competitive F_2 layer depending on the past learning of the adaptive weights of the bottom-up connections from F_1 to F_2. The signals are sent back from the winning unit in the F_2 layer down to F_1 via a top-down outstar network. The activation values produced in the units of F_1 due to this feedback are compared with the activation values due to input. If the two activation patterns match well, then the winning unit in the F_2 layer determines the category of the input pattern. If the match between activations due to top-down and input pattern is poor, as determined by a vigilance parameter, then the winning unit in F_2 does not represent the proper class for the input pattern **a**,. That unit is removed from the set of allowable winners in the F_2 layer. The other units in the F_2 layer are likewise skipped until a

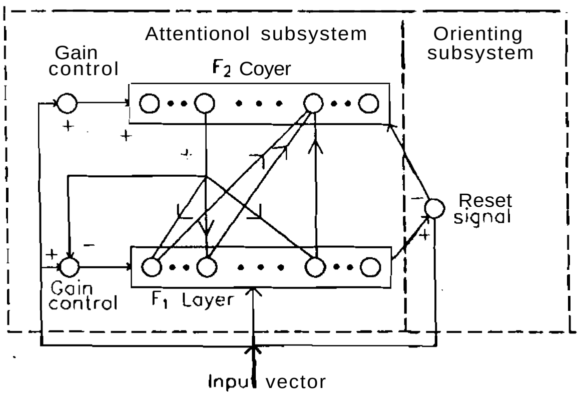


Figure 23. Adaptive resonance theory (ART) architecture. Two major subsystems are the attentional subsystem and the orienting subsystem. Units in each layer are fully interconnected to the units in the other layer.

suitable match is obtained between the activations in the F_1 layer due to top-down pattern and the input pattern. When a match is obtained, then both the bottom-up and top-down network weights are adjusted to reinforce the input pattern. If no match is obtained then an uncommitted (whose category is not identified during training) unit in the F_2 layer is committed to this input pattern, and the corresponding weights are adjusted to reinforce the input.

The above sequence of events conducts a search through the encoded patterns associated with each category trying to find a sufficiently close match with the input pattern. If no category exits, a new category is made. The search process is controlled by two subsystems, namely the orienting subsystem and the attentional subsystem. The orienting subsystem uses the dimensionless vigilance parameter that establishes the criterion for deciding whether the match is good enough to accept the input pattern as an exemplar of the chosen category. The gain control process in the attentional subsystem allows the units in F_1 to be engaged only when an input pattern is present, and it also actively regulates the learning (Freeman & Skupura **1991)**.

Stability is achieved in the ART network due to the dynamic matching and the control in learning. Plasticity is achieved in the ART due to its ability to commit an uncommitted unit in the F_2 layer for an input pattern belonging to a category different from what was already learnt.

ART gets its name from the particular way in which learning and recall interplay in the network. Information in the form of output signals from units reverberate back and forth between the two layers. If the proper patterns develop, a stable oscillation ensures, which is the neural network equivalent of resonance. During this resonance period learning or adjustment of adaptive weights takes place. Before the network has achieved a resonant state, no learning takes place, because the time required for changes in the weights is much longer than the time it takes the network to achieve resonance.

ART1 network was proposed to deal with binary input patterns (Carpenter & Grossberg **1988)**. ART2 network was developed to selforganize recognition categories for analog as well as binary input patterns (Carpenter & Grossberg **1987)**.

A minimal ART network can be embedded in a larger system to realize an associate memory. A system like CPN or multilayer perceptron directly maps pairs of patterns (a,,b,) during learning. If an ART system replaces the CPN, the resulting system becomes self stabilizing. Two ART systems can be used to pair sequences of the categories selforganized by the input sequences as shown in figure 24. The pattern recall can

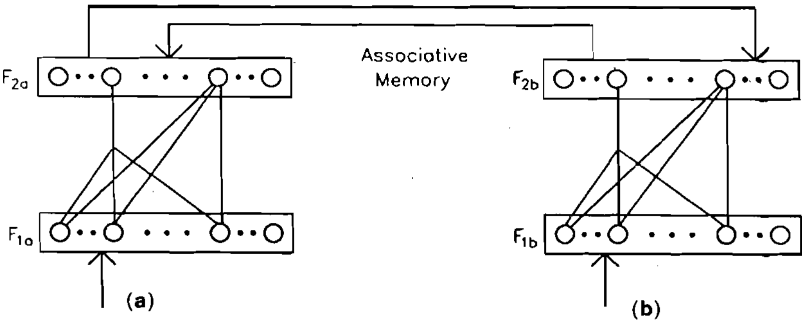


Figure 24. Two ART system combined to form an associative memory architecture.

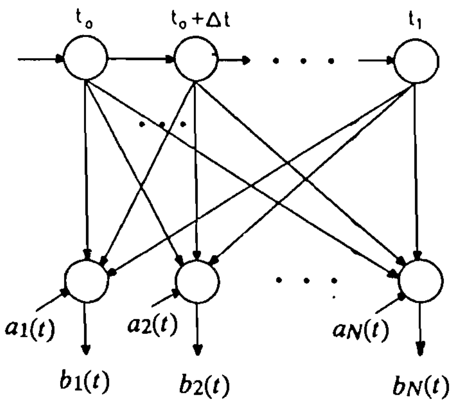


Figure 25. Grossberg's formal avalanche architecture.

occur in either direction during performance as in BAM. This scheme brings to the associate memory paradigm the code compression capabilities, as well as the stability properties of ART (Carpenter 1989).

5.4 Spatio-temporal patterns: temporal features – Avalanche

The ANN architectures described so far are applicable for recognition of patterns on the basis of information contained within the pattern itself. Even if a sequence of patterns with temporal correlation are presented, the previous or subsequent patterns have no effect on the classification of the current input pattern. But there are many applications (for example, speech recognition) where it is necessary to encode the information relating to the time correlation of spatial patterns, as well as the spatial pattern information itself.

Architectures for classification of spatio-temporal patterns (STP) are based on the Grossberg formal avalanche structure (Grossberg 1969). The structure (figure 25) of the network resembles the top two layers of the CPN, and both use multiple outstars. The avalanche architecture shows how a complex spatio-temporal pattern can be learned and recalled. Assume $\mathbf{a}(t) = (a_1(t), a_2(t), \ldots a_N(t))$ the spatial pattern required at time t. The sequence of $\mathbf{a}(t)$ at time intervals of At in the range $t_0 \leqslant t \leqslant t_1$ correspond to the desired spatio-temporal pattern. Activate the node labelled t_0 and apply $\mathbf{a}(t_0)$ to be learned by the outstar's output units. The second pattern a(t + At) is applied while activating the second **outstar,** labelled t_0 + At. Continue this process by activating successive **outstars** until all the patterns have been learned in sequence. Replay of the learned sequence can be initialized by stimulating the t_0 node, while a zero vector is applied to the a inputs. The output sequence $\mathbf{b}(t) \approx \mathbf{a}(t)$, for $t_0 \leqslant t \leqslant t_1$, is the learned sequence.

5.5 Pattern variability: recognition of deformed patterns – Neocognitron

Visual pattern recognition, such as recognition of handwritten characters or hand-drawn figures, is done effortlessly by human beings despite variability of features in different realizations of the pattern of the same character or figure. The patterns considered in the architectures described so far assume that the objects in the training and test patterns are identical in size, shape and position, except that in some cases there may be some noise added or some portions of the pattern missing. **Models of** associative memory can recover complete patterns from such imperfections, but normally cannot work if there is variability or deformation in the patterns of the test input.

Neural network models based on our understanding of human visual pattern recognition tend to perform well even for shifted and deformed patterns. In the visual area of the cerebrum, neurons respond selectively to local features of a visual pattern such as lines and edges. In areas higher than the visual cortex, cells exist that respond selectively to certain figures like circles, triangles, squares, human faces etc (Fukushima 1975). Thus the human visual system seems to have a hierarchical structure in which simple features are first extracted from the stimulus pattern, then integrated into more complicated ones. A cell at a higher stage generally receives signals from a wider area of the retina and is less sensitive to the position of the stimulus. Within the hierarchical structure of the visual systems are forward (afferent or bottom-up) and backward (efferent or top-down) propagation of signals. This kind of physiological evidence suggests a neural network structure for modelling the phenomenon of visual pattern recognition.

The objective is to synthesize a neural network model for pattern recognition for shifted and deformed patterns. The network model learns with a teacher (supervised learning) for reinforcement of the adaptive weights. The network model is called neocognitron. It is a hierarchical network (figure 26) consisting of many layers of cells, and has variable connections between cells in adjoining layers. It can be trained to recognize any set of patterns. After training, pattern recognition is performed on the basis of similarity in shape between patterns, and the recognition is not affected by deformation, or changes in size, or shifts in the positions of the input patterns (Fukushima 1988).

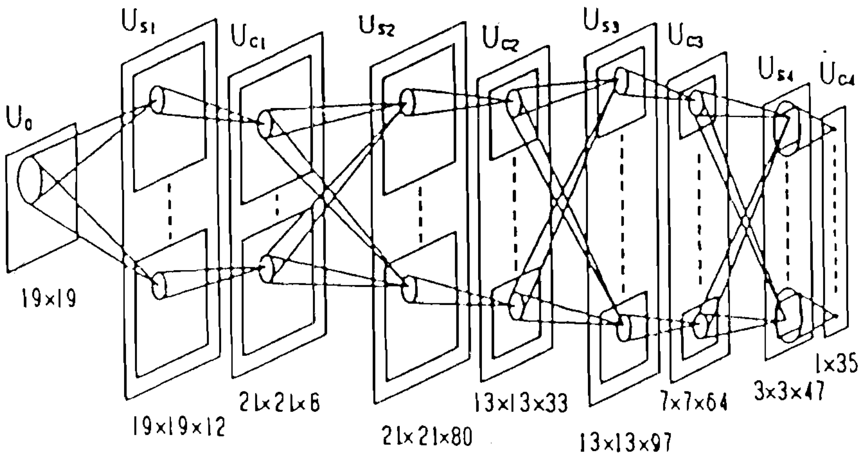


Figure 26. A hierarchical network structure of neocognitron (Fukushima 1991) for recognition of alphanumeric character recognition. The lowest stage of the network consists of a 2-dimensional array of receptor cells. Each succeeding stage has a layers consisting of S cells and C cells alternatively. Each layer is organized into groups of these cells, each group responding to a particular geometrical position. The numbers show the total numbers of S and C cells in individual layers of the network. S cells are feature extracting cells. The C cells are inserted to allow for positional errors in the feature.

In the hierarchical network of the neocognitron, local features of the input pattern are extracted by the cells of the lower stage, and they are gradually integrated into more global features. Finally, each cell of the highest stage integrates all the information of the input pattern, and responds only to one specific pattern. During the process of extracting and integrating features, errors in the relative positions of the local features are gradually tolerated. The operation of tolerating positional error a little at a time at each stage, rather than all in one step, plays an important role in endowing the network with the ability to recognize even distorted patterns (Fukushima **1991).**

Neocognitron also provides backward connections which will enable it to realize the selective attention feature of the visual pattern recognition. The selective attention feature relates to two or more patterns simultaneously present in the data, and our ability to focus on the desired one.

Neocognitron was developed for recognition of handwritten characters, although the ideas used in the architecture may be extended to other situations of pattern variability (Fukushima **1991).**

6. Applications

In this section of the paper we briefly discuss the application potential of neural network models and some research issues that are being currently addressed in this field. In applications we consider two different situations, one where the existing neural network concepts can be directly applied, and the other where there is potential for applying the neural network ideas but it is not yet clear how to formulate the real world problems to evolve a suitable neural network architecture. We will also list a few cases where neural network principles are being used in practice.

6.1 *Direct application*

In applications such as associative memories, optimization, vector quantization and pattern classification the principles of neural networks are directly applicable. In these applications it is assumed that the problem can be presented to the network directly, and what is being sought is the solution to the problem using the dynamics of the network. Many real world problems were formulated into one of these, and were solved successfully (Lisboa **1992).**

6.1a ***Associative memories (Bienenstock & von der Malsburg 1987; Hassoun 1989; Desai 1990; Kamp & Hasler 1990; Michel & Farrell 1990):*** As discussed earlier, the objective of associative memory is to store a pattern or data for later recall with partial or noisy version of the pattern as input, or to store association between two patterns for later recall of one of the patterns given the other. Both feedback and feedforward topologies of neural networks are directly used for these applications. Associative memory, if used in a feedback structure of the Hopfield type, can function as a content addressable memory as well. The stable states of the network, which represent the energy minima or basins of attraction, are used to store the pattern information. In a feedforward network the mapping function corresponding to the input–output pattern pairs is stored in the weights of the network.

Applications of these networks for associative memory is direct, if the patterns are available in the form of one or two dimensional array of values. Associative memories as content addressable memories are quite powerful. For example, if information about individuals is stored in a network, then it is possible to retrieve the complete data by providing partial or even noisy clues. Other common applications for an associative memory are recognition of images, and retrieval of bibliography information from partial references such as from incomplete title of a paper.

6.1b ***Optimization:*** One of the most successful applications of neural network principles is in solving optimization problems (Hopfield & Tank 1985; Kennedy & Chua 1988; Rauch & Winarske 1988; Tagliarini & Page 1988, pp. 775–82, Maa ***et al*** 1990, pp. 482–5). There are many situations where the problem can be formulated as minimization or maximization of a cost function or object function subjected to certain constraints. It is possible to map such a problem onto a feedback network, where the units and connection strengths are identified by comparing the cost function of the problem with the energy function of the network expressed in terms of the unit state values and the strengths of the connections. The solution to the problem lies in determining the state of the network at the global minimum of the energy function. In this process it is necessary to overcome the local minima of the energy function. This is accomplished by adopting a simulated annealing schedule for implementing the search for global minimum.

Probably the most studied problem in the context of optimization using principles of neural networks is the travelling salesman problem, where the objective is to find the shortest route connecting all cities to be visited by a salesman. Other optimization problems that are attempted include the weighted matching problem, where a number of points must be pairwise connected such that the sum of lengths of all connections is as short as possible, and stereo vision matching in optical image processing (Hertz ***et al*** 1991). The method of simulated annealing has also been successfully employed to find the optimal arrangement of integrated electronic circuits on semiconductor chips (Kirkpatrick ***et al*** 1983).

6.1c ***Vector quantization:*** Vector quantization (VQ) typically encodes a large set of training data vectors into a small set of representative points, thus achieving a significant compression in the representation of data. Vector quantization has been shown to be useful in compressing data that arises in image processing, speech processing, facsimile transmission, and weather satellites (Kohonen 1988; Nasrabadi & King 1988; Naylor & Li 1988).

Formally, vector quantization maps arbitrary data vectors to a binary representation or a symbol. The mapping is from an N-dimensional vector space to a finite set of symbols M. Associated with each symbol $m \in M$ is a reproduction vector $\hat{\mathbf{x}}_m$. The encoding of the data vector $\mathbf{x}$ to the symbol m is the mapping in VQ. The collection of all possible reproduction vectors is called the codebook.

The design of a codebook is called training, and it can be implemented using neural network models. The learning vector quantization (LVQ) structure is one such network model. Several other models have been proposed, for example, Kohonen's self-organising feature maps, to construct VQ codebooks for speech applications, and for image coding (Kohonen 1989; Ahalt ***et al*** 1990).

6.1d ***Pattern classification:*** Pattern classification is the most direct among all applications of neural networks. In fact neural networks became very popular because of the ability of a multilayer feedforward neural network to form complex decision regions in the pattern space for classification (Gorman & Sejnowski 1988; Lippmann 1989). Many pattern recognition problems, especially character or other symbol recognition and vowel recognition, have been successfully implemented using multilayered networks (LeCun *et al* 1989; Pal & Mitra 1992). Note however that these networks are not directly applicable for situations where the patterns are deformed or modified due to transformations such as translation, rotation and scale change (Dotsenko 1988; Seibert & Waxman 1989).

6.2 *Application areas*

Neural network concepts and principles appear to have great potential for solving problems arising in practice. For most practical problems the solution by neural networks is not obvious. This is because the problems cannot be mapped directly onto an existing neural network architecture. In fact there are no principles guiding us to this mapping. There are many pattern recognition tasks in speech and vision which we seem to perform effortlessly, but we do not understand how we do so. For example, in speech, our auditory mechanism processes the signal directly in a manner suitable for later neural processing. On the other hand, to prepare input to an artificial neural network, the speech signal is normally processed in fixed frames of 10–20 ms to extract a fixed number of spectral or related parameters. In this process the temporal and spectral features with proper resolution needed for recognition may not have been captured. Moreover, there is as yet no neural network architecture which could perform the speech pattern recognition with the same effectiveness as human beings do. Similar comments apply to problems in visual pattern recognition also. Some of the other areas where human performance cannot be matched by existing neural architectures are in motor control and decision making.

Despite realization of these issues, there are several cases where neural principles have been used successfully. A few of them are listed below in different areas for illustration (Lisboa 1992).

6.2a *Speech processing:* Recognition of isolated utterances of characters in a speaker-independent mode over a telephone line has been demonstrated for directory enquiring application (Lang *et al* 1990; Cole *et al* 1992).

Medium-size (about 50 words) vocabulary speaker independent isolated word recognition using a partially connected network has been demonstrated to give equal or better performance compared to the conventional methods based on dynamic time warping (Bottou *et al* 1990).

Reliable discrimination of some stop consonants was demonstrated using time delay neural network architectures, and these ideas were extended to derive network architectures for syllable recognition (Waibel 1989).

Text-to-speech conversion with limited capabilities for English was demonstrated using multilayered feedforward neural networks (Sejnowski & Rosenberg 1987).

6.2b *Computer vision:* Recognition of hand-printed digits has been one of the most successful applications of neural networks (Krzyzak *et al* 1990). Satellite image data compression and enhancement of noisy images are some of the other useful applications (Hertz *et al* 1991; Raghu *et al* 1993).

Transformation invariant object recognition is one of the most difficult tasks,

It is possible to view research in ANN along the following directions:

(i) ***Problem level:*** Involves issues in mapping the real world problems as pattern processors. This may require good understanding of human information processing both from the psychological and the biological angle.

(ii) ***Basic level:*** It is necessary to evolve better models of neurons as processing elements, their interconnections, dynamics (activation and synaptic), learning laws although some impressive demonstration of neural network architectures are available for handwritten characters (Fukushima & Miyake 1982).

6.2c ***Robotics and control:*** Artificial vision for autonomous navigation, path planning with obstacle avoidance, and parallel computation of inverse dynamics are some of the applications of neural networks in robotics (Kung & Hwang 1989; Handleman ***et al*** 1990; Kuperstein & Wang 1990).

Operation guidance in blast furnace control and modelling nonlinearities in chemical process control are some of the applications of neural networks in control areas (Bhat & McAvoy 1989; Konishi ***et al*** 1990).

6.2d ***Automated inspection and monitoring:*** Explosive-detection in aircraft luggage, industrial quality control through visual inspection, forecasting for the utility industries, sonar signal identification and fault diagnosis for sensor failure in industrial plants are examples of the application of neural networks in inspection and monitoring situations (Shea & Lin 1989; Naidu ***et al*** 1990).

6.2e ***Medical applications:*** Medical diagnosis, noise filters for cardiac signals, image processing of ultrasonograms, and discrimination of signals for patient monitoring, have all been successfully implemented using networks (Reggia & Suttonn 1988; Scalia ***et al*** 1988).

6.2f ***Business and finance:*** Scheduling and inventory control application, bond rating and asset forecasting in the stock market, exchange-rate forecasting, credit scoring, and mortgage underwriting have all demonstrated the successful use of neural network principles in business and finance (Collins ***et al*** 1988; Dutta & Shekkar 1988; White 1988).

7. Summary and Trends

In this tutorial article we have discussed the need for exploring new computing models for pattern recognition tasks. The importance of distinction between pattern processing and data processing has been discussed. The promise of the architectures inspired by the functions of biological neural networks has been shown by tracing the significant developments in artificial neural networks over the past decade. We have discussed the basics of artificial neural networks in terms of models of neurons, learning laws, and topology. We have also discussed the types of pattern recognition problems that can be solved by simple architectures based on the principles of artificial neural networks. Complex pattern recognition tasks require specialized architectures. Some general architectures were discussed for tasks requiring to resolve stability–plasticity dilemma and for tasks involving pattern variability and temporal patterns.

The most important issue for solving practical problems using the principles of ANN is still in evolving a suitable architecture to solve a problem. This continues to dominate this research area. ANN research may have to expand its scope to take into account the fuzzy nature of real world data and reasoning, and the complex (unknown) processing performed by the human perceptual mechanism through biological neural networks.

Cohen M, Grossberg S 1983 Absolute stability of global pattern formation and parallel storage by competitive neural networks. ***IEEE Trans. Syst.. Man Cybern.*** SMC-13: 815–825

Cole R, Fanty M, Muthuswamy Y, Gopalakrishna M 1992 Speaker-independent recognition of spoken English letters. ***Proc. Int. Joint Conf. Neural Networks***, San Diego, CA

Collins E, Ghosh S, Scotfield C L 1988 An application of a multiple neural network learning system to emulation of mortgage underwriting judgements. ***IEEE Int. Conf. Neural Networks*** (Piscataway, NJ: *IEEE* Press) 2: 459–466

Cybenko G 1989 Continuous value neural networks with two hidden layers are sufficient

It is possible to view research in ANN along the following directions:

(i) ***Problem level:*** Involves issues in mapping the real world problems as pattern processors. This may require good understanding of human information processing both from the psychological and the biological angle.
(ii) ***Basic level:*** It is necessary to evolve better models of neurons as processing elements, their interconnections, dynamics (activation and synaptic), learning laws and recall procedures.
(iii) ***Functional level***: Involves development of basic structures which can solve a class of pattern recognition problems. These form building blocks for development of new architectures.
(iv) ***Architecture level:*** This requires ideas to evolve new architectures from known principles, components and structures to solve complex pattern recognition problems. It is possible that the problems may be tailored somewhat to suit the architectures.
(v) ***Application level:*** The objective is to solve a given practical problem using generally the principles of ANN but with ideas from other areas also like physics, signal processing etc.

This paper is mostly a consolidation of work reported by several researchers in the literature, some of which is cited in the references. The author has borrowed several ideas and illustrations from the references quoted in this paper.

The author would like to thank Mr M Babu for his assistance in preparing this paper and Dr H M Chouhan for his critical comments. The author also thanks the members of the Speech and Vision Laboratory for their interaction in the seminars on topics related to neural networks. Finally, this paper would not have come to this stage but for the initiative and interest shown by Prof. N Viswanadham of the Indian Institute of Science, Bangalore. The author is grateful to him for his encouragement.

References

Abu-Mostafa Y S, St. Jaques J M **1985** Information capacity of the Hopfield model. ***IEEE Trans. Inf. Theor.*** **31: 461–464**

Ackley D M, Hinton G E, Sejnowski T J **1985 A** learning algorithm for Boltzmann machines. ***Cogn. Sci.*** **9: 147–169**

Ahalt S C, Krishnamurthy A K, Chen P, Melton D E **1990** Competitive learning algorithms for vector quantization. ***Neural Networks*** **3: 277–290**

Aleksander I, Morton H **1990** ***An introduction to neural computing*** (London: Chapman and Hall)

Bhat N, McAvoy T **1989** Use of neural nets for dynamic modelling and control of chemical process systems. ***Proc. Am. Autom. Contr. Conf.,*** Pittsburgh, PA, pp. **1342–1348**

Bienenstock E, von der Malsburg Ch **1987** A neural network for the retrieval of superimposed connection patterns. ***Euro. Phys. Lett.*** **3: 1243–1249**

Bottou L, Soulie F F, Blanchet P, Lienard J S **1990** Speaker independent isolated digit recognition: multilayer perceptrons vs. dynamic time warping. ***Neural Networks*** **3:436–465**

Carpenter G A **1989** Neural network models for pattern recognition and associative memory. ***Neural networks*** **2: 138–152**

Carpenter G A, Grossberg S **1987** ART2: Self-organization of stable category recognition codes for analog input patterns. ***Appl. Opt.*** **26: 4919–4930**

Carpenter G A, Grossberg S **1988** The ART of adaptive pattern recognition by a self-organizing neural network. ***IEEE Comput.*** **21: 77–88**

Cohen M, Grossberg S 1983 Absolute stability of global pattern formation and parallel storage by competitive neural networks. ***IEEE Trans. Syst., Man Cybern.*** SMC-13: 815–825

Cole R, Fanty M, Muthuswamy Y, Gopalakrishna M 1992 Speaker-independent recognition of spoken English letters. ***Proc. Int. Joint Conf. Neural Networks,*** San Diego, CA

Collins E, Ghosh S, Scotfield C L 1988 An application of a multiple neural network learning system to emulation of mortgage underwritingjudgements. ***IEEE Int. Conf. Neural Networks*** (Piscataway, NJ: IEEE Press) 2: 459–466

Cybenko G 1989 Continuous value neural networks with two hidden layers are sufficient. ***Math. Control. Signal Syst.*** 2: 303–314

Desai M S 1990 ***Noisy pattern retrieval using associative memories.*** MSEE thesis, University of Louisville, Kentucky

Deuker J, Schwartz D, Wittner B, Solla S, Howard R, Jackel L, Hopfield J 1987 Large automatic learning, rule extraction, and generalization. ***Complex Syst.*** 1: 877–922

Dotsenko V S 1988 Neural networks: translation-, rotation- and scale invariant pattern recognition ***J. Phys.*** A21: L783–L787

Dutta S, Shekkar S 1988 Bond rating: a non-conservative application of neural networks. ***IEEE Int. Conf. Neural Networks*** (Piscataway, NJ: IEEE Press) 2: 443–450

Freeman J A, Skupura D M 1991 ***Neural network algorithms, applications and programming techniques*** (New York: Addison–Wesley)

Fukushima K 1975 Cognitron: A self-organizing multilayer neural network. ***Biol. Cybern.*** 20: 121–136

Fukushima K 1988 A neural network for visual pattern recognition. ***IEEE Comput.*** 21: 65–75

Fukushima K 1991 Handwritten alphanumeric character recognition by the neocognitron. ***IEEE Trans. Neural Networks*** 2: 355–365

Fukushima K, Miyake S 1982 Neocognitron: a new algorithm for pattern recognition tolerant of deformations and shifts in position. ***Pattern Recogn.*** 15: 455–469

Geman S, Geman D 1984 Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. ***IEEE Trans. Pattern Anal. Machine Intell.*** PAMI-6: 721–741

Gorman R, Sejnowski T 1988 Learned classification of sonar targets using a massively parallel network. ***IEEE Trans. Acoust. Speech Signal Process.*** 36: 1135–1140

Grossberg S 1969 Some networks that can learn, and reproduce any number of complicated space-time patterns. ***Int. J. Math. Mech.*** 19: 53–91

Grossberg S 1980 How does a brain build a cognitive code? ***Psychol. Rev.*** 87: 1–51

Grossberg S 1982 ***Studies of mind & brain*** (Boston: Reidel)

Grossberg S 1988 Nonlinear neural networks: Principles, mechanisms, and architecture. ***Neural networks*** 1: 17–61

Handleman D H, Lane S H, Gelfand J J 1990 Integrating neural networks and knowledge-based systems for intelligent robotic control. ***IEEE Control Syst. Mag.*** 10(3): 77–87

Hassoun M H 1989 Dynamic heteroassociative memories. ***Neural Networks*** 2: 275–287

Hebb D 1949 ***Organization of the behaviour*** (New York: Wiley)

Hecht-Nielson R 1987 Counterpropagation networks. ***Appl. Opt.*** 26: 4979–4984

Hecht-Nielson R 1990 ***Neurocomputing*** (Reading, MA: Addison–Wesley)

Hertz J, Krogh A, Richard G P 1991 ***Introduction to the theory of neural computation*** (New York: Addison–Wesley)

Hinton G E, Sejnowski T J 1986 Learning and relearning in Boltzmann machines. In ***Parallel distributed processing: Explorations in the microstructure of cognition*** (eds) D E Rumelhart, J L McClelland (Cambridge, MA: MIT Press) 1: 282–317

Hodgkin A L, Huxley A F 1952 A quantitative description of membrane current and its application to conduction and excitation in nerve. ***J. Physiol.*** 117: 500–544

Hopfield J J 1982 Neural networks and physical systems with emergent collective computational capabilities. ***Proc. Natl. Acad. Sci. (USA)*** 79: 2554–2558

Hopfield J J, Tank D W 1985 Neural computation of decisions in optimization problems. ***Biol. Cybern.*** 52: 141–154

Huang Z, Kuh A 1992 A combined self-organizing feature map and multilayer perceptron for isolated word recognition. ***IEEE Trans. Signal Process.*** 40: 2651–2657

Hush D R, Horne B G 1993 Progress in supervised neural networks. ***IEEE Signal Process. Mag.*** 10: 8–39

Kamp Y, Hasler M 1990 ***Recursive neural networks for associative memory*** (Chichester: John Wiley & Sons)

Kennedy M P, Chau L O 1988 Neural networks for nonlinear programming. ***IEEE Trans. Circuits Syst.*** CAS-35: 554–562

Kirkpatrick S, Gelatt C D Jr, Vecchi M P 1983 Optimization by simulated annealing. ***Science*** 220: 671–680

Kohonen T 1988 An introduction to neural computing. ***Neural Networks*** 1: 3–16

Kohonen T 1989 ***Self-organization and associative memory*** (3rd edn) (Berlin: Springer–Verlag)

Kohonen T 1990 The self-organizing map. ***Proc. IEEE*** 78: 1464–1480

Konishi M, Otsuka Y, Matsuda K, Tamura N, Fuki A, Kadoguchi K 1990 Application of a neural network to operation guidance in a blast furnace. ***3rd European Seminar on Neural Computing: The Marketplace,*** London

Kosko B 1988 Bidirectional associative memories. ***IEEE Trans. Syst., Man Cybern.*** 18: 49–60

Kosko B 1990 Unsupervised learning in noise. ***IEEE Trans. Neural Networks*** 1: 44–57

Kosko B 1992 ***Neural networks and fuzzy systems*** (Englewood Cliffs, NJ: Prentice–Hall)

Krzyzak A, Dali W, Yuen C Y 1990 Unconstrained handwritten character classification using modified back propagation model. In ***Frontiers in handwritting recognition*** (ed.) C Y Suen (Montreal: CENPARMI)

Kung S Y, Hwang J N 1989 Neural network architectures for robotic applications. ***IEEE Trans. Robotics Autom.*** 5: 641–657

Kuperstein M, Wang J 1990 Neural controller for adaptive movements with unforeseen payloads. ***IEEE Trans. Neural Networks*** 1(1): 137–142

Lang K J, Waibel A H, Hinton G E 1990 A time-delay neural network architecture for isolated word recognition. ***Neural Networks*** 3(1): 23–44

LeCun Y, Boser B, Denker J S, Henderson D, Howard R E, Hubbard W, Jackel L D 1989 Back propagation applied to handwritten zip code recognition. ***Neural Comput.*** 1: 541–551

Lippmann R P 1987 An introduction to computing with neural nets. ***IEEE Trans. Acoust. Speech Signal Process. Mag.*** (April): 4–22

Lippmann R P 1989a Review of neural networks for speech recognition. ***Neural Comput.*** 1(1): 1–38

Lippmann R P 1989b Pattern classification using neural networks. ***IEEE Commun. Mag.*** (Nov): 47–64

Lisboa P G P 1992 ***Neural networks current applications*** (London: Chapman & Hall)

Maa C Y, Chin C, Shanblatt M A 1990 A constrained optimization neural net techniques for economic power dispatch. ***Proc.*** 1990 (New York: IEEE Press)

Marcus A, van Dam A 1991 User-interface developments for the nineties. ***IEEE Comput.*** 24: 49–57

McCulloch W S, Pitts W 1943 A logical calculus of the ideas immanent in nervous activity. ***Bull. Math. Biophys.*** 5: 115–133

Michel A N, Farrell J A 1990 Associative memories via artificial neural networks. ***IEEE Control Syst. Mag.*** (April): 6–17

Minsky M, Papert S A 1988 ***Perceptron*** (Cambridge, MA: MIT Press)

Muller B, Reinhardt J 1990 ***Neural networks: An introduction*** (Berlin: Springer–Verlag)

Murakami K, Aibara T 1987 An improvement on the Moore–Penrose generalized inverse associative memory. ***IEEE Trans. Syst. Man. Cybern.*** SMC 17: 699–706

Naidu S R, Zafiriou E, McAvoy T J 1990 Use of neural networks for sensor failure detection in a control system. ***IEEE Control Syst. Mag.*** 10(3): 49–55

Nasrabadi N M, King R A 1988 Image coding using vector quantization: A review. ***IEEE Trans. Commun.*** 36: 957–971

Naylor J, Li K P 1988 Analysis of a neural network algorithm for vector quantization of speech parameters. ***Neural Networks*** 1 (Suppl): 310

Pal S K, Mitra S 1992 Multilayer perceptron, fuzzy sets, and classification. ***IEEE Trans. Neural Networks*** 3: 683–697

Raghu P P, Chouhan H M, Yegnanarayana B 1993 Multispectral image classification using neural network. ***Proc. Natl. Conf. on Neural Networks,*** Anna University, Madras (NCNN): 1–10

Rauch H E, Winarske T 1988 Neural networks for routing communications traffic. ***IEEE Control Syst. Mag.*** (April): 26–31

Reggia J A, Suttonn G G 1988 III. Self-processing networks and their biomedical implications. ***Proc. IEEE*** 76: 680–692

Rosenblatt F 1958 A probabilistic model for information storage and organization in the brain. ***Psychol. Rev.*** 65: 386–408

Rosenblatt F 1962 ***Principles of neurodynamics*** (Washington, DC: Spartan)

Rumelhart D, McClelland J 1986 ***Parallel distributed processing: Explorations in the microstructure of cognition*** (Boston: MIT Press) vol. 1

Rumelhart D E, Zipser D 1986 Feature discovery by competitive learning. ***Parallel and distributed processing*** (eds) J L McClelland, D E Rumelhart 1: 151–193

Scalia F, Marconi L, Ridella S, Arrigo P, Mansi C, Mela G S 1988 An example of back propagation: diagnosis of dyspepsia. ***Ist IEE Conf. Neural Networks*** (IEE Conf. Publ.) 313: 332–540

Schalkoft R 1992 ***Pattern recognition – Statistical, structural and neural approaches*** (New York: John Wiley & Sons)

Seibert M, Waxman A 1989 Spreading activation layers, visual saccades, and invariant representations for neural pattern recognition systems. ***Neural Networks*** 2: 9–27

Sejnowski T, Rosenberg C 1987 Parallel networks that learn to pronounce English text. ***Complex Syst.*** 1: 145–168

Shea P M, Lin V 1989 Detection of explosives in checked airline baggage using an artificial neural system. ***Int. Joint. Conf. on Neural Networks*** 2: 31–34

Simpson K P 1990 ***Artificial neural systems*** (New York: Pergamon)

Simpson K P 1992 ***Foundations of neural networks in artificial neural networks*** (eds) Edgar Sanchez-Sinencio, Clifford Lau (New York: IEEE Press)

Szu H 1986 Fast simulated annealing. In ***Neural networks for computing*** (ed.) J S Denker (New York: Snowbird)

Tagliarini G A, Page E W 1988 A neural network solution to the concentrator assignment problem. ***Neural information processing systems*** (ed.) *D* Z Anderson (New York: Am. Inst. Phys.)

von der Malsburg Ch 1973 Self-organization of orientation sensitive cells in the striate cortex. ***Kybernetik*** 14: 85–100

Waibel A 1989 Modular construction of time-delay neural networks for speech recognition. . ***Neural Comput.*** 1: 39–46

Wasserman P D 1988 Combined backpropagation/cauchy machine. ***Neural networks: Abstracts of the first INNS Meeting,*** Boston (Elmsford, NY: Pergamon) 1: 556

White H 1988 Economic prediction using neural networks: the case of IBM daily stock returns. ***Neural networks: Abstracts of the First INNS Meeting,*** Boston (Elmsford, *NY:* Pergamon) 1: 451–458

Widrow B, Hoff M E 1960 Adaptive switching circuits. ***IRE WESCON Convention Record*** (4): 96–104

Willshaw D J, von der Malsburg Ch 1976 How patterned neural connections can be set up by self-organization. ***Proc. R. Soc. London*** B194: 431–445

Zurada J M 1992 ***Introduction to artificial neural systems*** (St. Paul, MN: West)