

Artificial Neural Networks: A Tutorial

Anil K. Jain* Jianchang Mao⁺ K. Mohiuddin⁺

*Computer Science Department ⁺IBM Almaden Research Center

Michigan State University

650 Harry Road

East Lansing, MI 48824

San Jose, CA 95120

Abstract

Numerous advances have been made in developing “intelligent” programs, some of which have been inspired by biological neural networks. Researchers from various scientific disciplines are designing artificial neural networks (ANNs) to solve a variety of problems in decision making, optimization, prediction, and control. Artificial neural networks can be viewed as parallel and distributed processing systems which consist of a huge number of simple and massively connected processors. There has been a resurgence of interest in the field of ANNs in recent years. This article intends to serve as a tutorial for those readers with little or no knowledge about ANNs to enable them to understand the remaining articles of this special issue. We discuss the motivations behind developing ANNs, main issues of network architecture and learning process, and basic network models. We also briefly describe one of the most successful applications of ANNs, namely automatic character recognition.

1 Introduction

What are artificial neural networks (ANNs)? Why is there so much excitement about ANNs? What are the basic models used in designing ANNs? What tasks can ANNs perform efficiently? These are the main questions addressed in this tutorial article.

Let us first consider the following classes of challenging problems of interest to computer scientists and engineers.

Pattern classification: The task of pattern classification is to assign an input pattern (e.g., speech waveform or handwritten symbol) represented by a feature vector to one of pre-specified classes (Fig. 1(a)). Well-known applications of pattern classification are character recognition, speech recognition, EEG waveform classification, blood cell classification, and printed circuit board inspection.

Clustering/categorization: In clustering, also known as unsupervised pattern classification, there are no training data with known class labels. A clustering algorithm explores the similarity between the patterns and places *similar* patterns in a cluster (see Fig. 1(b)). Well-known clustering applications include data mining, data compression, and exploratory data analysis.

Function approximation: Given a set of n labeled training patterns (input-output pairs), $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$, generated from an unknown function $\mu(\mathbf{x})$ (subject to noise), the task of function approximation is to find an estimate, say $\hat{\mu}$, of the unknown function μ (Fig. 1(c)). Various engineering and scientific modeling problems require function approximation.

Prediction/forecasting: Given a set of n samples $\{y(t_1), y(t_2), \dots, y(t_n)\}$ in a time sequence, t_1, t_2, \dots, t_n , the task is to predict the sample $y(t_{n+1})$ at some future time t_{n+1} . Prediction/forecasting has a significant impact on decision making in business, science and engineering. Stock market prediction and weather forecasting are typical applications of prediction/forecasting techniques (see Fig. 1(d)).

Optimization: A wide variety of problems in mathematics, statistics, engineering, science, medicine, and economics can be posed as optimization problems. The goal of an optimization algorithm is to find a solution satisfying a set of constraints such that an objective function is maximized or minimized. A classical optimization problem is the Traveling Salesperson Problem (TSP), which is an *NP-complete* problem.

Content-addressable memory: In the Von Neumann model of computation, an entry in memory is accessed only through its address which is independent of the content in the memory. Moreover, if a small error is made in calculating the address, a completely different item would be retrieved. Associative memory or content-addressable memory, as the name implies, can be accessed by its content. The content in the memory can be recalled even by a

partial input or distorted content (see Fig. 1(f)). Associative memory is extremely desirable in building multimedia information databases.

Control: Consider a dynamic system defined by a tuple $\{u(t), y(t)\}$, where $u(t)$ is the control input and $y(t)$ is the resulting output of the system at time t . In *model-reference adaptive control*, the goal is to generate a control input $u(t)$ such that the system follows a desired trajectory determined by the reference model. An example of model reference adaptive control is the engine idle speed control (Fig. 1(g)).

A large number of approaches have been proposed for solving the problems described above. While successful applications of these approaches can be found in certain well-constrained environments, none of them is flexible enough to perform well outside the domain for which it is designed. The field of artificial neural networks has provided alternative approaches for solving these problems. It has been established [1, 8, 6] that a large number of applications can benefit from the use of ANNs.

Artificial neural networks, which are also referred to as *neural computation*, *network computation*, *connectionist models*, and *parallel distributed processing* (PDP), are massively parallel computing systems consisting of an extremely large number of simple processors with many interconnections between them.

The purpose of this article is to serve as a tutorial for those readers with little or no knowledge about artificial neural networks. The rest of this article is organized as follows. Section 2 provides the motivations behind developing ANNs. In Section 3, we describe the basic neuron model, network architecture, and learning process. Sections 4 through 7 provide more details about several well-known ANN models: multilayer feedforward networks, Kohonen's self-organizing maps, ART models and the Hopfield network. In Section 8, we discuss character recognition, a popular and one of the most successful applications of ANN models. Concluding remarks are presented in Section 9.

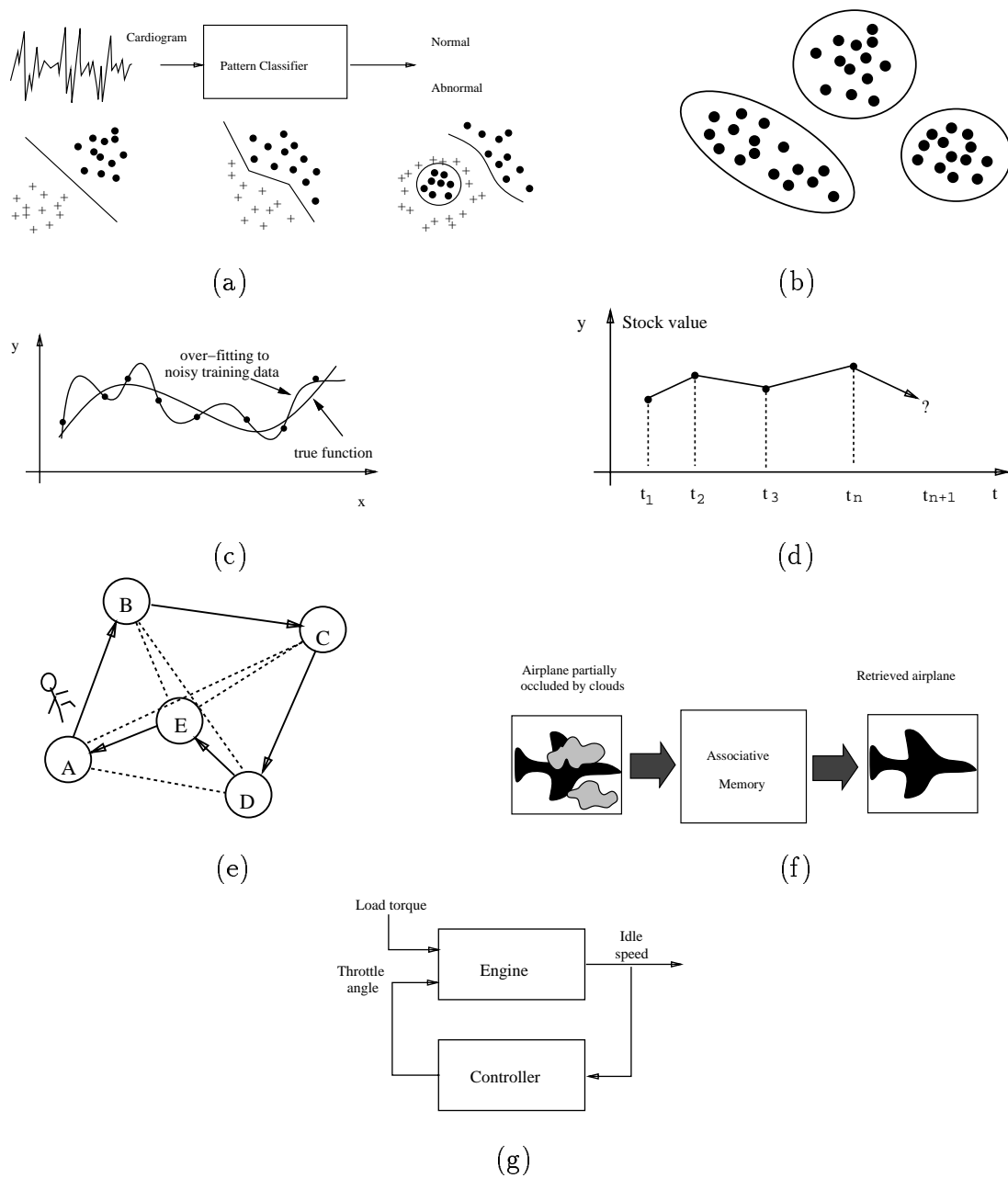


Figure 1: Tasks that neural networks can perform. (a) Pattern classification; (b) clustering/categorization; (c) Function approximation; (d) Prediction/forecasting; (e) Optimization (TSP problem); (f) Retrieval by content; and (g) Engine idle speed control.

2 Motivation

ANNs are inspired by biological neural networks. This section provides a brief introduction to biological neural networks.

2.1 Biological Neural Networks

A *neuron* (or nerve cell) is a special biological cell with information processing ability. A

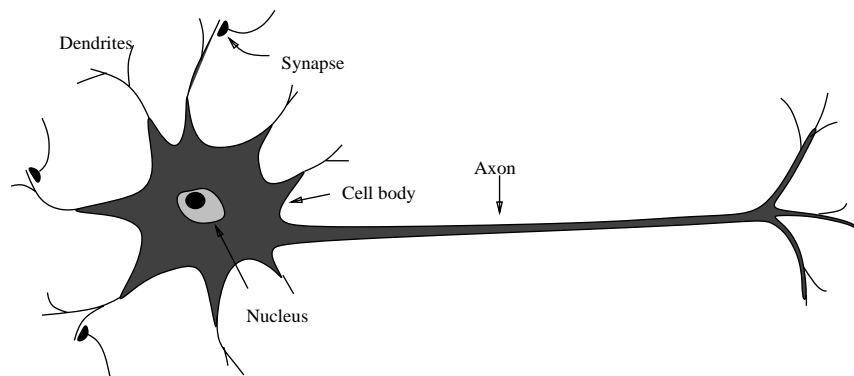


Figure 2: A sketch of a biological neuron.

schematic drawing of a neuron is shown in Fig. 2. A neuron is composed of a cell body, or *soma*, and two types of out-reaching tree-like branches: *axon* and *dendrites*. The cell body has a nucleus which contains information on hereditary traits and a plasma containing molecular equipment for the production of material needed by the neuron. A neuron receives signals (impulses) from other neurons through its dendrites (receivers), and transmits signals generated by its cell body along the axon (transmitter) which eventually branches into strands and substrands. At the terminals of these strands are the *synapses*. A synapse is a place of contact between two neurons (an axon strand of one neuron and a dendrite of another neuron). When the impulse reaches the synapse's terminal, certain chemicals, called neurotransmitters are released. The neurotransmitters diffuse across the synaptic gap, and their effect is to either enhance or inhibit, depending on the type of the synapse, the receptor neuron's own tendency to emit electrical impulses. The effectiveness of a synapse can be adjusted by the signals passing through it so that the synapses can *learn* from the activities in

which they participate. This dependence on past history acts as a memory which is possibly responsible for the human ability to remember.

The cerebral cortex in humans is a large flat sheet of neurons about 2 to 3 mm thick with a surface area of about 2,200 cm^2 , about twice the area of a standard computer keyboard. The cerebral cortex contains about 10^{11} neurons, which is approximately the number of stars in the Milky Way! Neurons are massively connected, much more complex and denser than today's telephone networks. Each neuron is connected to $10^3 - 10^4$ other neurons. In total, the human brain contains approximately $10^{14} - 10^{15}$ interconnections.

Neurons communicate by a very short train of pulses, typically milliseconds in duration. The *message* is modulated on the frequency with which the pulses are transmitted. The frequency can vary from a few up to several hundred Hertz, which is a million times slower than the fastest switching speed in electronic circuits. However, complex perceptual decisions, such as face recognition, are made by a human very quickly, typically within a few hundred milliseconds. These decisions are made by a network of neurons whose operational speed is only a few milliseconds. This implies that computations involved cannot take more than about one hundred serial stages. In other words, the brain runs parallel programs that are about 100 steps long for such perceptual tasks. This is known as the *hundred step rule* [5]. The same timing considerations show that the amount of information sent from one neuron to another must be very small (a few bits). This implies that critical information is not transmitted directly, but captured and distributed in the interconnections, and hence the name *connectionist* model.

Interested readers can find more introductory and easily comprehensible material on biological neurons and neural networks in [3].

2.2 Why Artificial Neural Networks?

Modern digital computers have outperformed humans in the domain of numeric computation and related symbol manipulation. However, humans can effortlessly solve complex perceptual problems (e.g., recognizing a person in a crowd from a mere glimpse of his face) at such a high speed and extent as to dwarf the world's fastest computer. Why does there exist such a remarkable difference in their performance? The biological computer employs a completely

different architecture than the Von Neumann architecture (see Table 1). It is this difference that significantly affects the type of functions each computational model is best able to perform.

	Von Neumann computer	Biological computer
Processor	complex	simple
	high speed	low speed
	one or a few	a large number
Memory	separate from a processor	integrated into processor
	localized	distributed
	non-content addressable	content addressable
Computing	centralized	distributed
	sequential	parallel
	stored programs	self-learning
Reliability	very vulnerable	robust
Expertise	numerical and symbolic manipulations	perceptual problems
Operating environment	well-defined, well-constrained	poorly-defined, unconstrained

Table 1: Von Neumann computer versus biological computer.

Numerous efforts have been made to develop “intelligent” programs based on Von Neumann’s centralized architecture. However, such efforts have not resulted in any general-purpose intelligent programs. ANN models are inspired by biological evidence, and attempt to make use of some of the “organizational” principles that are believed to be used in the human brain. Our ability to model a biological nervous system using ANNs can increase our understanding of biological functions. The state-of-the-art in computer hardware technology (e.g., VLSI and optical) has made such modeling feasible.

The long course of evolution has resulted in the human brain possessing many desirable characteristics which are neither present in a Von Neumann computer nor in modern paral-

lel computers. These characteristics include *massive* parallelism, distributed representation and computation, learning ability, generalization ability, adaptivity, inherent contextual information processing, fault tolerance, and low energy consumption. It is hoped that ANNs, motivated from biological neural networks, would possess some of these desirable characteristics.

The field of artificial neural networks is an interdisciplinary area of research. A thorough study of artificial neural networks requires a knowledge about neurophysiology, cognitive science/psychology, physics (statistical mechanics), control theory, computer science, artificial intelligence, statistics/mathematics, pattern recognition, computer vision, parallel processing, and hardware (digital/analog/VLSI/optical). New developments in these disciplines continuously nourish the field of ANNs. On the other hand, artificial neural networks also provide an impetus to these disciplines in the form of new tools and representations. This symbiosis is necessary for the vitality of neural network research. Communications among these disciplines ought to be encouraged.

2.3 Brief Historical Review

Research in ANNs has experienced three consecutive cycles of enthusiasm and skepticism. The first peak, dating back to the 1940's, is due to McCullough and Pitt's pioneering work [14]. The second period of intense activity occurred in the 1960's which featured Rosenblatt's perceptron convergence theorem [18] and Minsky and Papert's work showing the limitations of a simple perceptron [16]. Minsky and Papert's results dampened the enthusiasm of most researchers, especially those in the computer science community. As a result, there was a lull in the neural network research for almost 20 years. Since the early 1980's, ANNs have received considerable renewed interest. The major developments behind this resurgence include Hopfield's energy approach [9] in 1982, and the backpropagation learning algorithm for multilayer perceptrons (multilayer feedforward networks) which was first proposed by Werbos [20], reinvented several times, and popularized by Rumelhart et al. [19] in 1986. Anderson and Rosenfeld [2] provide a detailed historical account of developments in ANNs.

3 Artificial Neural Networks

This section provides an overview of ANNs. First, computational models of neurons are introduced. Then, the important issues of network architecture and learning are discussed. Various ANN models are organized by their architecture and the learning algorithm involved.

3.1 Computational Models of Neurons

McCulloch and Pitts [14] proposed a binary threshold unit as a computational model for a neuron. A schematic diagram of a McCulloch-Pitts neuron is shown in Fig. 3. This

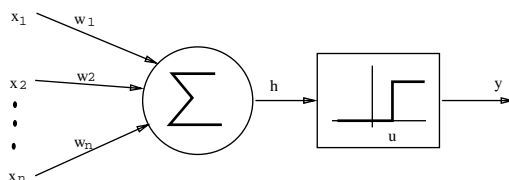


Figure 3: McCulloch-Pitts model of a neuron.

mathematical neuron computes a weighted sum of its n input signals, x_j , $j = 1, 2, \dots, n$, and generates an output of “1” if this sum is above a certain threshold u , and an output of “0” otherwise. Mathematically,

$$y = \theta \left(\sum_{j=1}^n w_j x_j - u \right),$$

where $\theta(\cdot)$ is a unit step function at zero, and w_j is the synapse weight associated with the j^{th} input. For simplicity of notation, we often consider the threshold u as another weight $w_0 = -u$ which is attached to the neuron with a constant input, $x_0 = 1$. Positive weights correspond to *excitatory* synapses, while negative weights model *inhibitory* synapses. McCulloch and Pitts proved that with suitably chosen weights a synchronous arrangement of such neurons is, in principle, capable of universal computation. There is a crude analogy here to a biological neuron: wires and interconnections model axons and dendrites, connection weights represent synapses, and the threshold function approximates the activity in soma. The model of McCulloch and Pitts contains a number of simplifying assumptions, which do not reflect the true behavior of biological neurons.

The McCulloch-Pitts neuron has been generalized in many ways. An obvious generalization is to use activation functions other than the threshold function, e.g., a piecewise linear, *sigmoid*, or Gaussian, shown in Fig. 4. The sigmoid function is by far the most frequently used function in ANNs. It is a strictly increasing function that exhibits smoothness and has the desired asymptotic properties. The standard sigmoid function is the *logistic* function, defined by

$$g(x) = 1/(1 + \exp(-\beta x)),$$

where β is the slope parameter.

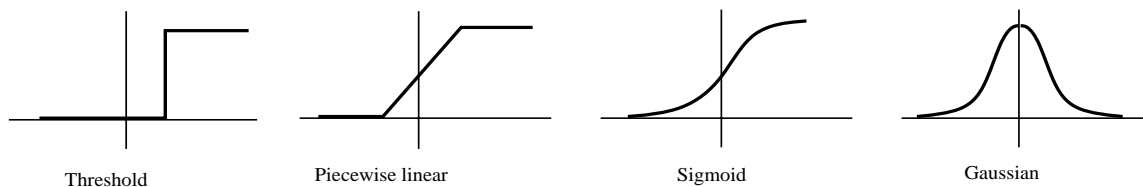


Figure 4: Different types of activation functions.

3.2 Network Architecture

An assembly of artificial neurons is called an artificial neural network. ANNs can be viewed as weighted directed graphs in which nodes are artificial neurons and directed edges (with weights) are connections from the outputs of neurons to the inputs of neurons. Based on the connection pattern (architecture), ANNs can be grouped into two major categories as shown in Fig. 5: (i) *feedforward* networks in which no loop exists in the graph, and (ii) *feedback* (or *recurrent*) networks in which loops exist because of feedback connections. The most common family of feedforward networks is a layered network in which neurons are organized into layers with connections strictly in one direction from one layer to another. Fig. 5 also shows typical networks of each category. We will discuss in this article all these networks except for the Radial Basis Function (RBF) networks [6] which employ the same network architecture as multilayer perceptrons, but different activation functions.

Different connectivities yield different network behaviors. Generally speaking, feedforward networks are *static* networks, i.e., given an input, they produce only one set of output

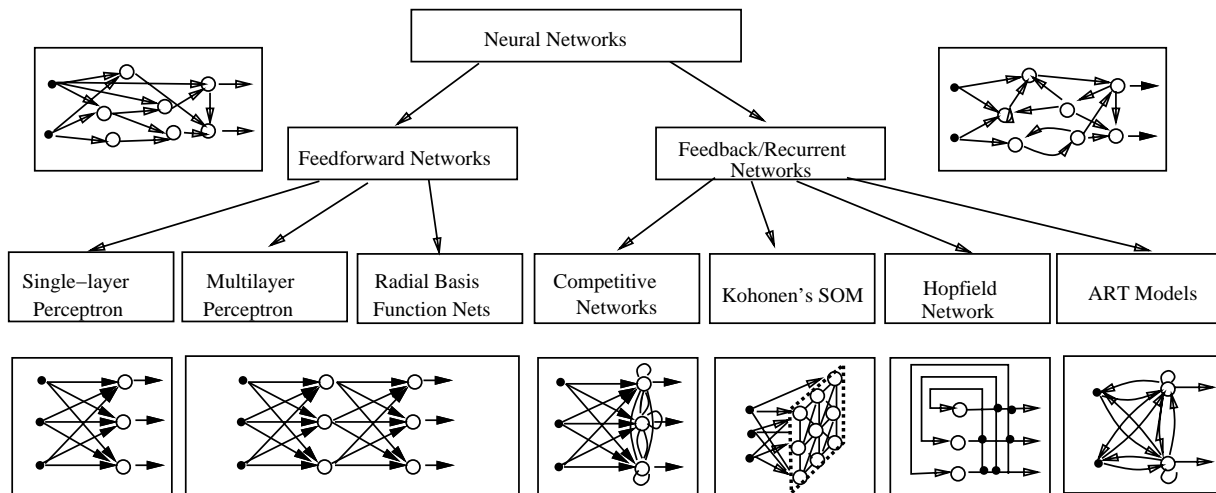


Figure 5: A taxonomy of network architectures.

values, not a sequence of values. Feedforward networks are memoryless in the sense that the response of a feedforward network to an input is independent of the previous state of the network. Recurrent networks are dynamic systems. Upon presenting a new input pattern, the outputs of the neurons are computed. Because of the feedback paths, the inputs to each neuron are then modified, which leads the network to enter a new state.

Different network architectures require different learning algorithms. The next section will provide a general overview of the various learning processes.

3.3 Learning

The ability to learn is a fundamental trait of intelligence. Although a precise definition of learning is often difficult to state, a learning process in the context of artificial neural networks can be viewed as the problem of updating network architecture and connection weights so that a network can efficiently perform a specific task. Most of the time, the network must learn the connection weights from the available training patterns. Improvement in performance is achieved over time through iteratively updating the weights in the network. The ability of artificial neural networks to automatically *learn from examples* makes them very attractive and exciting. Instead of having to specify a set of *rules*, ANNs appear to learn them from the given collection of representative examples. This is one of the major

advantages of neural networks over traditional expert systems.

In order to understand or design a learning process, one must first have a model of the environment in which a neural network operates, i.e., what information is available to the neural network. We refer to this model as a learning paradigm [6]. Second, one must understand how weights in the network are updated, i.e., what are the *learning rules* which govern the updating process. A *learning algorithm* refers to a procedure in which learning rules are used for adjusting weights in the network.

There are three main learning paradigms, namely, (i) supervised, (ii) unsupervised, and (iii) hybrid learning. In *supervised learning* or *learning with a teacher*, the network is provided with a correct answer to every input pattern. Weights are determined so that the network can produce answers as close as possible to the known correct answers. *Reinforcement* learning is a variant of supervised learning where the network is provided with only a critique on the correctness of network outputs, not the correct answers (outputs) themselves. In contrast, *unsupervised learning* or *learning without a teacher* does not require any correct answer associated with each input pattern in the training data set. It explores the underlying structure in the data, or correlations between patterns in the data, and organizes patterns into categories from these correlations. *Hybrid learning* combines supervised learning and unsupervised learning. Typically, a portion of weights in the network are determined using supervised learning, while the others are obtained from unsupervised learning.

Learning theory must address three fundamental and practical issues associated with learning from samples: (i) capacity, (ii) sample complexity, and (iii) time complexity. *Capacity* concerns how many patterns can be stored, and what functions and decision boundaries can be formed by a network.

Sample complexity determines the number of training patterns needed to train the network in order to guarantee a valid generalization. Too few patterns may cause “over-fitting” (wherein the network performs well on the training data set, but poorly on independent test patterns drawn from the same distribution as the training patterns) (see Fig. 1(c)).

Computational complexity refers to the time requirement for a learning algorithm to estimate a solution from the training patterns. Many existing learning algorithms have high computational complexity. Designing efficient algorithms for neural network learning is a

very active research topic.

There are four basic types of learning rules: (i) error-correction, (ii) Boltzmann, (iii) Hebbian, and (iv) competitive learning.

3.3.1 Error-Correction Rules

In the supervised learning paradigm, the network is given a desired output for each input pattern. During the learning process, the actual output, y , generated by the network may not equal the desired output, d . The basic principle of error-correction learning rules is to use the error signal $(d - y)$ to modify the connection weights such that this error will be gradually reduced.

The well-known perceptron learning rule is based on this error-correction principle. A perceptron consists of a single neuron with adjustable weights, w_j , $j = 1, 2, \dots, n$, and threshold μ , as shown in Fig. 3. Given an input vector $\mathbf{x} = (x_1, x_2, \dots, x_n)^t$, the net input to the neuron (before applying the threshold function) is

$$v = \sum_{j=1}^n w_j x_j - \mu.$$

The output y of the perceptron is $+1$ if $v > 0$, and 0 otherwise. In a two-class classification problem, the perceptron assigns an input pattern to one class if $y = 1$, and to the other class if $y = 0$. The linear equation

$$\sum_{j=1}^n w_j x_j - \mu = 0,$$

defines the decision boundary (a hyperplane in the n -dimensional input space) which divides the space into two halves.

Rosenblatt [18] developed a learning procedure to determine the weights and threshold in a perceptron, given a set of training patterns. The perceptron learning procedure can be

described as follows.

1. Initialize the weights and threshold to small random numbers.
2. Present a pattern vector $(x_1, x_2, \dots, x_n)^t$, and evaluate the output of the neuron.
3. Update the weights according to

$$w_j(t+1) = w_j(t) + \eta(d - y)x_j,$$

where d is the desired output, t is iteration number, and η ($0.0 < \eta < 1.0$) is the gain (step size).

Note that learning occurs only when an error is made by the perceptron. Rosenblatt proved that if the training patterns are drawn from two linearly-separable classes, then the perceptron learning procedure will converge after a finite number of iterations. This is the well known *perceptron convergence theorem*. In practice, one does not know whether the patterns are linearly separable or not. Many variations of this learning algorithm have been proposed in the literature [8]. Other activation functions can also be used, which lead to different learning characteristics. However, *a single layer perceptron can only separate linearly separable patterns, as long as a monotonic activation function is used.*

The well-known backpropagation learning algorithm (described in section 4) is also based on the error-correction principle.

3.3.2 Boltzmann Learning

Boltzmann machines are symmetric recurrent networks consisting of binary units (+1 for “on” and -1 for “off”). By symmetric, we mean that the weight on the connection from unit i to unit j is equal to the weight on the connection from unit j to unit i ($w_{ij} = w_{ji}$). Only a portion of neurons, *visible* neurons, interact with the environment, the rest, called hidden neurons, do not interact. Each neuron is a stochastic unit which generates an output (or state) according to the Boltzmann distribution of statistical mechanics. Boltzmann machines operate in two modes: (i) *Clamped* mode in which visible neurons are clamped onto specific states determined by the environment; and (ii) *Free-running* mode in which both the visible and hidden neurons are allowed to operate freely.

Boltzmann learning is a stochastic learning rule derived from information-theoretic and thermodynamic principles (see [2]). The objective of Boltzmann learning is to adjust the

connection weights such that the states of visible units satisfy a particular desired probability distribution. According to the Boltzmann learning rule, the change in the connection weight w_{ij} is given by

$$\Delta w_{ij} = \eta(\bar{\rho}_{ij} - \rho_{ij}),$$

where η is the learning rate, and $\bar{\rho}_{ij}$ and ρ_{ij} are the correlations between the states of unit i and unit j when the network operates in the clamped mode and free-running mode, respectively. The values of $\bar{\rho}_{ij}$ and ρ_{ij} are usually estimated from *Monte Carlo* experiments which are extremely slow.

Boltzmann learning can be viewed as a special case of error-correction learning in which error is measured not as the direct difference between the desired output and actual output, but as the difference between the correlations between the outputs of two neurons under two operating conditions (clamped and free-running).

3.3.3 Hebbian Rule

The oldest learning rule is *Hebb's postulate of learning* [7]. It was proposed by Hebb based on the following observation from neurobiological experiments: if neurons on both sides of a synapse are activated synchronously and repeatedly, then the strength of that synapse is selectively increased [6].

Mathematically, the Hebbian rule can be described as

$$w_{ij}(t+1) = w_{ij}(t) + \eta y_j(t)x_i(t),$$

where x_i and y_j are the output values of neurons i and j , respectively, which are connected by the synapse w_{ij} , and η is the learning rate. Note that x_i is the input to the synapse.

An important property of this rule is that learning is done locally, i.e., the change of the synapse weight depends only on the activities of the two neurons connected by it. This significantly simplifies the complexity of the learning circuit in a VLSI implementation.

A single neuron trained using the Hebbian rule exhibits an orientation selectivity. Fig. 6 demonstrates this property. The points depicted in Fig. 6 are drawn from a 2-dimensional Gaussian distribution and used for training a neuron. The weight vector of the neuron is initialized to \mathbf{w}_0 as shown in the figure. As the learning proceeds, the weight vector moves

closer and closer to the direction \mathbf{w} of maximal variance in the data. In fact, \mathbf{w} is the eigenvector of the covariance matrix of the data corresponding to the largest eigenvalue.

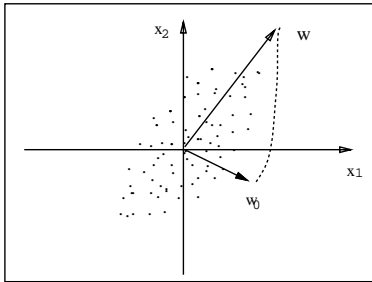


Figure 6: Orientation selectivity of a single neuron trained using the Hebbian rule.

3.3.4 Competitive Learning Rules

Unlike Hebbian learning (where multiple output units can be fired simultaneously), competitive learning has all the output units compete among themselves for activation. As a result of such a competition, only one output unit, is active at any given time. This phenomenon is often known as *winner-take-all*. Competitive learning has been found to exist in biological neural networks [6].

The outcome of competitive learning is often a clustering or categorization of the input data. Similar patterns are grouped by the network and represented by a single unit. This grouping process is done by the network automatically based on the correlations in the data.

The simplest competitive learning network consists of a single layer of output units as shown in Fig. 5. Each output unit i in the network connects to all the input units (x_i 's) via weights, w_{ij} , $j = 1, 2, \dots, d$. Each output unit also connects to all the other output units via inhibitory weights, but has a self-feedback with an excitatory weight. As a result of competition, only the unit i^* with the largest (or the smallest) net input becomes the winner, i.e., $\mathbf{w}_{i^*} \cdot \mathbf{x} \geq \mathbf{w}_i \cdot \mathbf{x}$, $\forall i$, or $\|\mathbf{w}_{i^*} - \mathbf{x}\| \leq \|\mathbf{w}_i - \mathbf{x}\|$, $\forall i$. When all the weight vectors are normalized, these two inequalities are equivalent.

A simple competitive learning rule can be stated as follows.

$$\Delta w_{ij} = \begin{cases} \eta(x_j^t - w_{i^*j}), & i = i^*, \\ 0, & i \neq i^*. \end{cases} \quad (1)$$

Note that only the weights of the winner unit get updated. The effect of this learning rule is to move the stored pattern in the winner unit (weights) a little bit closer to the input pattern. A geometric interpretation of competitive learning is demonstrated in Fig. 7. In this example, we assume that all the input vectors have been normalized to have unit length. They are depicted as black dots in Fig. 7(a). The weight vectors of the three units are randomly initialized. Their initial positions and final positions on the sphere after competitive learning are shown as crosses in Figs. 7(a) and 7(b), respectively. As we can see from Fig. 7, each of the three natural groups (clusters) of patterns has been discovered by an output unit whose weight vector points to the center of gravity of the discovered group.

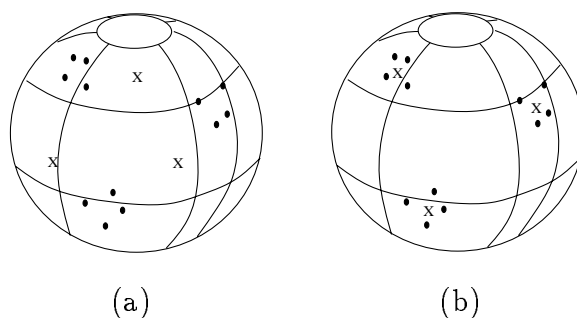


Figure 7: An example of competitive learning: (a) before learning; (b) after learning.

One can see from the competitive learning rule that the network will never stop learning (updating weights) unless the learning rate η is zero. It is possible that a particular input pattern may fire different output units at different iterations during learning. This brings up the stability issue of a learning system. A learning system is said to be *stable* if no pattern in the training data changes its category after a finite number of learning iterations. One way of achieving stability is to force the learning rate to decrease gradually as the learning process proceeds, and so it eventually approaches zero. However, this artificial freezing of learning causes another problem termed *plasticity*, which is defined as the ability to adapt to new data. This is known as Grossberg's *stability-plasticity* dilemma in competitive learning.

The most well-known example of competitive learning is *vector quantization* for data compression. Vector quantization has been widely used in speech and image processing for efficient storage, transmission and modeling. The goal of vector quantization is to represent a set or distribution of input vectors by a relatively small number of prototype vectors (weight

vectors), or a codebook. Once a codebook has been constructed and agreed upon, we need only transmit or store the index of the corresponding prototype to the input vector. Given an input vector, its corresponding prototype can be found through searching for the nearest prototype in the codebook.

3.3.5 Summary of Learning Algorithms

Various learning algorithms and their associated network architectures are summarized in Table 2. However, this is by no means an exhaustive list of the learning algorithms available in the literature. Both supervised and unsupervised learning paradigms employ learning rules based on error-correction, Hebbian, and competitive learning. Learning rules based on error-correction can be used for training feedforward networks, while Hebbian learning rules have been used for all types of network architectures. However, each learning algorithm is designed for training a specific network architecture. Therefore, when we talk about a learning algorithm, it is implied that there is a particular network architecture associated with it. Each learning algorithm is able to perform well on at most a few tasks. The last column of table 2 lists a number of tasks that each learning algorithm can perform. Due to space limitations, we will not discuss some of the other algorithms, including ADALINE, MADALINE [13], linear discriminant analysis [10], ART2, ARTMAP [4], Sammon's projection [10], and principal component analysis [8]. Interested readers can further consult the corresponding references (in order to reduce the size of the bibliography, this article does not always cite the first paper that proposed a particular algorithm).

4 Multilayer Feedforward Networks

Fig. 8 shows a typical 3-layer perceptron. In general, a standard L -layer feedforward network¹ consists of one input stage, $(L - 1)$ hidden layers, and one output layer of units which are successively connected (fully or locally) in a feedforward fashion with no connections between units in the same layer and no feedback connections between layers.

The most popular class of multi-layer feedforward networks is *multi-layer perceptrons* in

¹In this paper, we adopt the convention that the input nodes are not counted as a layer.

Paradigm	Learning Rule	Architecture	Learning Algorithm	Task
Supervised	Error-correction	Single- or Multi-layer Perceptron	Perceptron learning algorithms Backpropagation ADALINE & MADALINE	pattern classification function approximation prediction, control
	Boltzmann	Recurrent	Boltzmann learning algorithm	pattern classification
	Hebbian	Multi-layer Feedforward	Linear discriminant analysis	data analysis pattern classification
		Competitive	Competitive	Learning vector quantization
	ART network	ARTMAP	pattern classification within-class categorization	
Unsupervised	Error-correction	Multi-layer Feedforward	Sammon's projection	data analysis
	Hebbian or Competitive	Feedforward	Principal component analysis	data analysis data compression
		Hopfield Net	Associative memory learning	associative memory
	Competitive	Competitive	Vector quantization	categorization data compression
		Kohonen SOM	Kohonen's SOM	categorization data analysis
		ART networks	ART1, ART2	categorization
Hybrid	Error-correction & Competitive	RBF network	RBF learning algorithm	pattern classification function approximation prediction, control

Table 2: Well-known learning algorithms.

which each computational unit employs either the thresholding function or the sigmoid function. Multi-layer perceptrons are capable of forming arbitrarily complex decision boundaries and can represent any Boolean function [16]. The development of the *back-propagation* learning algorithm for determining weights in a multi-layer perceptron has made these networks the most popular among researchers as well as users of neural networks.

We denote $w_{ij}^{(l)}$ as the weight on connection between the i^{th} unit in layer $(l - 1)$ to j^{th} unit in layer l .

Let $\{(\mathbf{x}^{(1)}, \mathbf{d}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{d}^{(2)}), \dots, (\mathbf{x}^{(p)}, \mathbf{d}^{(p)})\}$ be a set of p training patterns (input-output pairs), where $\mathbf{x}^{(i)} \in R^n$ is the input vector in the n -dimensional pattern space, and $\mathbf{d}^{(i)} \in [0, 1]^m$, a m -dimensional hyper-cube. For classification purposes, m is the number of classes. The squared-error cost function, which is most frequently used in the ANN literature, is defined as

$$E = \frac{1}{2} \sum_{i=1}^p \|\mathbf{y}^{(i)} - \mathbf{d}^{(i)}\|^2. \quad (2)$$

The back-propagation algorithm [19] is a gradient-descent method to minimize the squared-

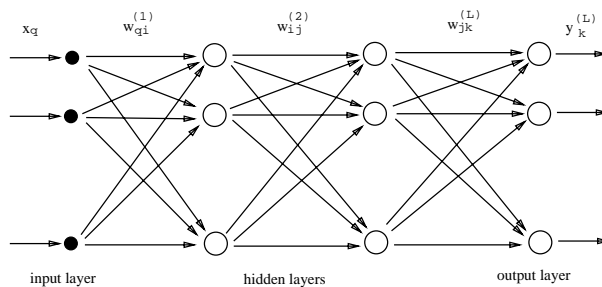


Figure 8: A typical 3-layer feedforward network architecture.

error cost function in Equation (2), and is given below.

1. Initialize the weights to small random values;
2. Randomly choose an input pattern $\mathbf{x}^{(\mu)}$;
3. Propagate the signal forward through the network;
4. Compute δ_i^L in the output layer ($o_i = y_i^L$)

$$\delta_i^L = g'(h_i^L)[d_i^\mu - y_i^L],$$

where h_i^l represents the *net input* to the i^{th} unit in the l^{th} layer, and g' is the derivative of the activation function g .

5. Compute the deltas for the preceding layers by propagating the errors backwards;

$$\delta_i^l = g'(h_i^l) \sum_j w_{ij}^{l+1} \delta_j^{l+1},$$

for $l = (L - 1), \dots, 1$.

6. Update weights using

$$\Delta w_{ji}^l = \eta \delta_i^l y_j^{l-1}$$

7. Go to step 2 and repeat for the next pattern until the error in the output layer is below a pre-specified threshold or a maximum number of iterations is reached.

A geometric interpretation (adopted and modified from [13]) shown in Fig. 9 can help explicate the role of hidden units (with the threshold activation function). Each unit in the first hidden layer forms a hyper-plane in the pattern space; boundaries between pattern classes can be approximated by hyper-planes. A unit in the second hidden layer forms a

hyper-region from the outputs of the first-layer units; a decision region is obtained by performing an “AND” operation on hyperplanes. The output-layer units combine the decision regions made by the units in the second hidden layer by performing logical “OR” operations. Remember that this scenario is depicted only to help us understand the role of hidden units. Their actual behavior, after we train the network, could be different from this. A two-layer network can form more complex decision boundaries than what is depicted in Fig. 9. Moreover, multilayer perceptrons with sigmoid activation functions can form smooth decision boundaries rather than piece-wise linear boundaries.


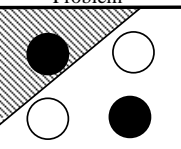

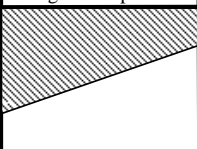
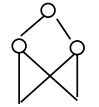
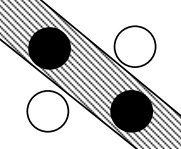

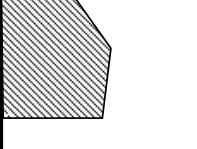
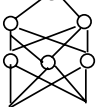
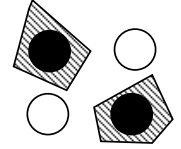

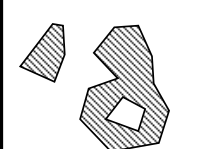
Structure	Description of Decision Regions	Exclusive-OR Problem	Classes with Meshed Regions	General Region Shapes
 Single layer	Half plane bounded by hyperplane			
 Two-layer	Arbitrary (Complexity limited by number of hidden units)			
 Three-layer	Arbitrary (Complexity limited by number of hidden units)			

Figure 9: A geometric interpretation of the role of hidden units.

A special class of multi-layer feedforward networks is the *Radial Basis Function (RBF)* network [6], a two-layer network. Each unit in the hidden layer employs a radial basis function, such as a Gaussian kernel, as the activation function. The radial basis function (or kernel function) is centered at the point specified by the weight vector, associated with the unit. Both the positions and the widths of these kernels must be learned from the training patterns. The number of kernels in the RBF network is usually much less than the number of training patterns. Each output unit implements a linear combination of these radial basis functions. From the point of view of function approximation, the hidden units provide a set of functions that constitute an arbitrary “basis” for representing input patterns in the space

spanned by the hidden units.

There are a variety of learning algorithms for the RBF network [6]. The basic algorithm employs a two-step learning strategy (hybrid learning): estimation of kernel positions and kernel widths using some unsupervised clustering algorithm, followed by a supervised least mean square (LMS) type of algorithm to determine the connection weights to the output layer. Since the output units are linear, a non-iterative algorithm can be used. After this initial solution is obtained, a supervised gradient-based algorithm can be used to refine the network parameters.

This hybrid learning algorithm for training the RBF network converges much faster than the backpropagation algorithm for training multi-layer perceptrons. However, for many problems, the RBF network often involves a larger number of hidden units compared with a multi-layer perceptron. This implies that the run-time (after training) speed of the RBF network is often slower than the run-time speed of a multi-layer perceptron. The efficiencies (error versus network size) of the RBF network and the multi-layer perceptron are, however, problem-dependent. It has been shown that the RBF network has the same asymptotic approximation power as a multi-layer perceptron.

There are many issues in designing feedforward networks. These issues include: (i) how many layers are needed for a given task?; (ii) how many units per layer?; (iii) what can we expect a network to generalize on data not included in the training set?; and (iv) how large should the training set be for “good” generalization? Although multilayer feedforward networks using backpropagation have been widely used for classification and function approximation (see [8]), many design parameters still have to be determined by trial-and-error. Existing theoretical results provide only very loose guidelines for selecting these parameters in practice.

5 Kohonen’s Self-Organizing Maps

Kohonen’s self-organizing map (SOM) [11] has the desirable property of topology preservation which captures an important aspect of the feature maps in the cortex of the more developed animal brains. By a topology preserving mapping, we mean that nearby input patterns should activate nearby output units on the map. The basic network architecture

of Kohonen's SOM is shown in Fig. 5. It basically consists of a two-dimensional array of units, each of which is connected to all d input nodes. Let \mathbf{w}_{ij} denote the d -dimensional vector associated with the unit at location (i, j) of the 2-D array. Each neuron computes the Euclidean distance between the input vector \mathbf{x} and the stored weight vector \mathbf{w}_{ij} .

Kohonen's SOM is a special type of competitive learning network which defines a spatial neighborhood for each output unit. The shape of the local neighborhood can be either square, rectangular, or circular. Initial neighborhood size is often set to $1/2$ to $2/3$ of the network size, and shrinks with time according to some schedule (e.g., an exponentially decreasing function). During competitive learning, all the weight vectors associated with the winner and its neighboring units are updated.

Kohonen's SOM learning algorithm can be described as follows.

1. Initialize weights to small random numbers; set initial learning rate and neighborhood;
2. Present a pattern \mathbf{x} , and evaluate the network outputs;
3. Select the unit (c_i, c_j) with the minimum output:

$$\|\mathbf{x} - \mathbf{w}_{c_i c_j}\| = \min_{ij} \|\mathbf{x} - \mathbf{w}_{ij}\|$$

4. Update all the weights according to the following learning rule;

$$\mathbf{w}_{ij}(t+1) = \begin{cases} \mathbf{w}_{ij}(t) + \alpha(t)[\mathbf{x}(t) - \mathbf{w}_{ij}(t)], & \text{if } (i, j) \in N_{c_i c_j}(t), \\ \mathbf{w}_{ij}(t), & \text{otherwise,} \end{cases}$$

where $N_{c_i c_j}(t)$ is the neighborhood of the unit (c_i, c_j) at time t , and $\alpha(t)$ is the learning rate.

5. Decrease the value of $\alpha(t)$ and shrink the neighborhood $N_{c_i c_j}(t)$;
6. Repeat steps 2 – 5 until the change in weight values is less than a pre-specified threshold, or a maximum number of iterations is reached.

Kohonen's SOM can be used for projection of multivariate data, density approximation, and clustering. Some successful applications of Kohonen's SOM can be found in the areas of speech recognition, image processing, robotics, and process control [8]. The design parameters include the dimensionality of the neuron array, number of neurons in each dimension, shape of neighborhood, shrinking schedule of the neighborhood, and the learning rate.

6 Adaptive Resonance Theory Models

Recall that an important issue in competitive learning is the *stability-plasticity* dilemma. How do we learn new things (plasticity) and yet retain the stability which ensures that the existing knowledge is not erased or corrupted? Carpenter and Grossberg's Adaptive Resonance Theory models (ART1, ART2, and ARTMAP) were developed in an attempt to overcome this dilemma [4]. The basic idea of these models is as follows. The network has a sufficient supply of output units, but they are not used until deemed necessary. A unit is said to be *committed* (*uncommitted*) if it is (is not) being used. The learning algorithm updates the stored prototypes of a category only if the input vector is sufficiently similar to them. An input vector and a stored prototype are said to resonate when they are sufficiently similar. The extent of similarity is controlled by a *vigilance parameter*, ρ , with $0 < \rho < 1$, which also determines the number of categories. When the input vector is not sufficiently similar to any existing prototype in the network, a new category is created and an uncommitted unit is assigned to this new category with the input vector as the initial prototype. If no such uncommitted unit exists, then a novel input generates no response.

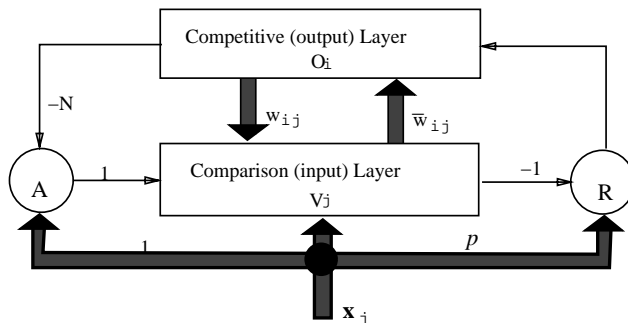


Figure 10: ART1 network.

We present only ART1 which takes binary (0/1) input to illustrate the model. Fig. 10 shows a simplified diagram of the ART1 architecture (see [8]). It consists of two layers of units which are fully connected. A top-down weight vector \mathbf{w}_j is associated with unit j in the input layer, and bottom-up weight vector $\bar{\mathbf{w}}_i$ is associated with output unit i ; $\bar{\mathbf{w}}_i$ is the

normalized version of \mathbf{w}_i .

$$\bar{\mathbf{w}}_i = \frac{\mathbf{w}_i}{\varepsilon + \sum_j w_{ji}}, \quad (3)$$

where ε is a small number which is used to break the ties in selecting the winner. The top-down weight vectors, \mathbf{w}_j 's, store the prototypes of clusters. The role of normalization is to prevent prototypes with a long vector length from dominating prototypes with a short vector length. Given an N -bit input vector \mathbf{x} , the output of the auxiliary unit A is given by

$$A = Sgn_{0/1}\left(\sum_j x_j - N \sum_i O_i - 0.5\right),$$

where $Sgn_{0/1}(x)$ is the *signum* function which produces $+1$ if $x \geq 0$ and 0 otherwise, and the output of an input unit is given by

$$\begin{aligned} V_j &= Sgn_{0/1}\left(x_j + \sum_i w_{ji}O_i + A - 1.5\right) \\ &= \begin{cases} x_j, & \text{if no output } O_j \text{ is "on",} \\ x_j \wedge \sum_i w_{ji}O_i, & \text{otherwise.} \end{cases} \end{aligned}$$

A reset signal R is generated only when the similarity is less than the vigilance level.

The ART1 learning algorithm is described below.

1. Initialize $w_{ij} = 1$, for all i, j . Enable all the output units.
2. Present a new pattern \mathbf{x} .
3. Find the winner unit i^* among the enabled output units

$$\bar{\mathbf{w}}_{i^*} \cdot \mathbf{x} \geq \bar{\mathbf{w}}_i \cdot \mathbf{x}, \quad \forall i$$

4. Vigilance test

$$r = \frac{\mathbf{w}_{i^*} \cdot \mathbf{x}}{\sum_j x_j}.$$

If $r \geq \rho$ (resonance), goto Step 5. Otherwise, disable unit i^* and goto Step 3 (until all the output units are disabled).

5. Update the winning weight vector \mathbf{w}_{i^*} , enable all the output units and goto Step 2

$$\Delta w_{ji^*} = \eta(V_j - w_{ji^*}).$$

6. If all the output units are disabled, select one of the uncommitted output units and set its weight vector to x . If there is no uncommitted output unit (capacity is reached), the network rejects the input pattern.

The ART1 model is able to create new categories and to reject an input pattern when the network reaches its capacity. However, the number of categories discovered in the input data by ART1 is sensitive to the vigilance parameter.

7 Hopfield Network

The Hopfield network uses a network *energy* function as a tool for designing recurrent networks and for understanding its dynamic behavior [9]. Hopfield's formulation made explicit the principle of storing information as dynamically stable attractors, and popularized the use of recurrent networks for associative memory and for solving combinatorial optimization problems.

A Hopfield network with N units has two versions: binary and continuous valued networks. Let v_i be the state or output of the i^{th} unit. For binary networks, v_i is either +1 or -1, but for continuous networks, v_i can be any value between 0 and 1. Let w_{ij} be the synapse weight on the connection from unit i to unit j . In Hopfield network, $w_{ij} = w_{ji}$, $\forall i, j$ (symmetric network), and $w_{ii} = 0$, $\forall i$ (no self-feedback connections). The network dynamics for the binary Hopfield network is

$$v_i = \text{Sgn}\left(\sum_j w_{ij}v_j - \theta_i\right). \quad (4)$$

The dynamic update of network states in Equation (4) can be carried out in at least two ways: *synchronously* and *asynchronously*. In a synchronous updating scheme, all the units are updated simultaneously at each time step. A central clock is therefore required to synchronize the process. On the other hand, an asynchronous updating scheme selects one unit at a time, and updates its state. The unit for updating can be chosen randomly.

The energy function of the binary Hopfield network in a state $\mathbf{v} = (v_1, v_2, \dots, v_N)^T$ is given by

$$E = -\frac{1}{2} \sum_i \sum_j w_{ij}v_i v_j. \quad (5)$$

The central property of the energy function is that as the state of network evolves according to the network dynamics (Eq. (4)), the network energy always decreases, and eventually reaches a local minimum point (attractor) where the network stays with a constant energy.

Suppose a set of patterns is stored in these attractors of a network. Then it can be used as an *associative memory*. Any pattern present in the basin of attraction of a stored pattern can be used as an index to retrieve it.

An associative memory usually operates in two phases: storage and retrieval. In the storage phase, the weights in the network are determined in such a way that the attractors of the network memorize a set of p N -dimensional patterns $\{\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^p\}$ to be stored. A generalization of the Hebbian learning rule can be used for setting connection weights w_{ij} . In the retrieval phase, the input pattern is used as the initial state of the network, and the network evolves according to the network dynamics. A pattern is produced (or retrieved) when the network reaches an equilibrium state.

How many patterns can be stored in a network with N binary units? In other words, what is the *memory capacity* of a network? Note that the capacity is finite because a network with N binary units has a maximum of 2^N distinct states, and not all of them are attractors. Moreover, not all the attractors (stable states) can store useful patterns. There also exist *spurious attractors* which store patterns different from any of the patterns in the training set [8].

It has been shown that the maximum number of random patterns that a Hopfield network can store is $P_{max} \approx 0.15N$. If the number of stored patterns $p < 0.15N$, then a nearly perfect recall can be achieved. If memory patterns are orthogonal vectors instead of random patterns, then more patterns can be stored. But, the number of spurious attractors increases as p reaches the capacity limit. Several learning rules have been proposed for increasing the memory capacity of Hopfield networks (see [8]). Note that we require N^2 connections in the network to store p N -bit patterns.

Hopfield networks always evolve in the direction that leads to a lower network energy. This implies that if a combinatorial optimization problem can be formulated as minimizing the network energy, then the Hopfield network can be used to find the optimal (or suboptimal) solution by letting the network evolve freely. In fact, any quadratic objective function can be rewritten in the form of Hopfield network energy. For example, the classical traveling salesperson problem can be formulated as a network energy minimization problem.

8 Applications

We have discussed a number of important ANN models and learning algorithms proposed in the literature. These ANN models and learning algorithms have been widely used for solving the seven classes of problems that are described in Section 1. In Table 2, we show the typical tasks that each of the ANN models and learning algorithms is particularly suitable for. It is important to keep in mind that in order to successfully apply an ANN model and learning algorithm to a real-world problem, one must deal with a number of design issues, including network model, network size, activation function, learning parameters, and number of training samples. In this section, we take one of the most successful applications of ANNs, *Optical Character Recognition (OCR)*, as an example to illustrate how multilayer feedforward networks are used in practice.

OCR deals with the problem of processing a scanned image of text and transcribing it into a machine readable form. In this section we will outline the basic components of OCR and explain how ANNs are used for character classification.

An OCR system usually consists of the following modules: (i) preprocessing, (ii) segmentation, (iii) feature extraction, (iv) classification, and (v) contextual processing. A paper document is scanned to produce a gray level or binary (black-and-white) image. In the preprocessing stage, filtering is applied to remove noise, and text areas are located and converted to a binary image using either a global or a local adaptive thresholding method. In the segmentation step, the text image is separated into individual characters. This is a particularly difficult task with handwritten text where there is a proliferation of touching characters. One effective technique is to break the composite pattern into smaller patterns (over-segmentation) and find the correct character segmentation points using the output of a pattern classifier.

Recognizing segmented characters is not an easy task because there are many different writing styles, different degrees of slant, skew, and noise level. This is evident from Fig. 11 which shows the size-normalized character bitmaps of a sample set from the NIST hand-print character database [21].

There are two main schemes for using ANNs in an OCR system as shown in Fig. 12. The first scheme employs an explicit feature extractor (not necessarily a neural network).



Figure 11: A sample set of characters in the NIST data.

For instance, contour direction features are used in Fig. 12. The extracted features are passed to the input stage of a multilayer feedforward network (e.g., [17]). This scheme is very flexible in incorporating a large variety of features. The other scheme does not explicitly extract features from the raw data. The feature extraction implicitly takes place within the intermediate stages of the ANN. A nice property of this scheme is that feature extraction and classification are integrated and trained simultaneously to produce “optimal” classification results. It is not clear whether the types of features which can be extracted by this integrated architecture are the most effective ones for character recognition. Moreover, this scheme requires a much larger network than the first scheme.

A typical example of this integrated feature extraction-classification scheme is the network developed by Le Cun et al. [12] for zip-code recognition. A 16×16 normalized gray level image is presented to a feedforward network with three hidden layers. The units in the first hidden layer are locally connected to the units in input layer, forming a set of local feature maps. The second hidden layer is constructed in a similar way as the first hidden layer. Each unit in the second hidden layer also combines local information coming from feature maps in the first hidden layer. The activation level of an output unit can be interpreted as an approximation of the *a posteriori* probability of belonging to a particular class given the input pattern. The output categories are ordered according to activation levels and passed to the post-processing stage. In the post-processing stage, contextual information is

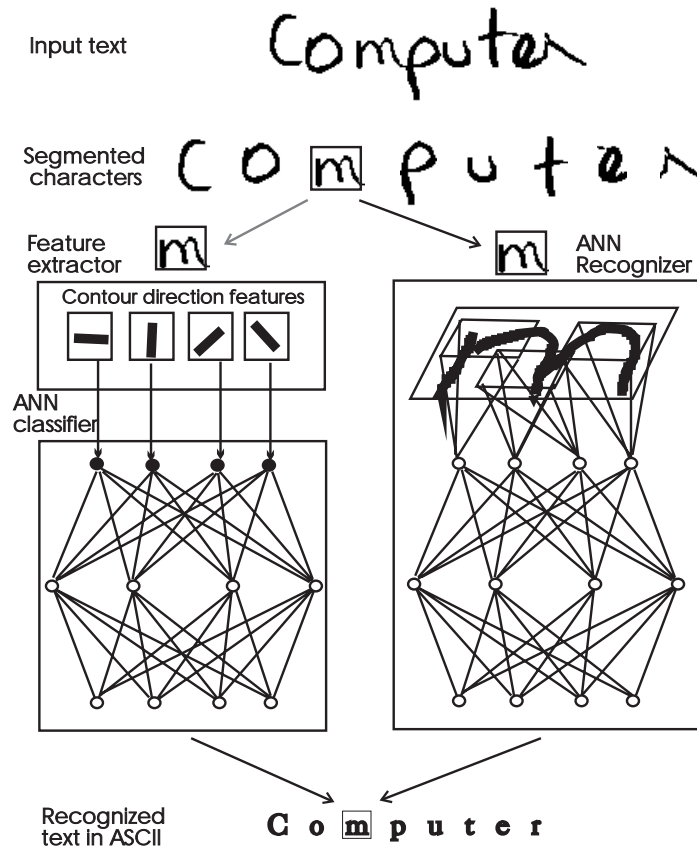


Figure 12: Two schemes for using ANNs in an OCR system.

exploited to update the output of the classifier. This could, for example, involve looking up a dictionary of admissible words, or utilizing syntactic constraints present, for example, in phone numbers or social security numbers.

How good are ANNs for OCR? ANNs are found to work very well in practice. However, there is no conclusive evidence about ANN's superiority over conventional statistical pattern classifiers. At the First Census Optical Character Recognition System Conference held in 1992 [21], more than 40 different handwritten character recognition systems were evaluated based on their performance on a common database. The top ten performers among them used either some type of multilayer feedforward network or a nearest neighbor-based classifier. ANNs tend to be superior in terms of speed and memory requirements compared to nearest neighbor methods. Unlike the nearest neighbor methods, classification speed using ANNs is independent of the size of the training set. The recognition accuracies of the top OCR

systems on the NIST isolated (pre-segmented) character data were above 98% for digits, 96% for upper-case characters, and 87% for lower-case characters. One conclusion drawn from the test is that the recognition performance of OCR systems on isolated characters is comparable to the human performance. However, humans still outperform OCR systems on unconstrained and cursive handwritten documents.

9 Concluding Remarks

Developments in ANNs have stimulated a lot of enthusiasm and criticism as well. Many comparative studies provide an optimistic outlook for ANNs, while others offer a pessimistic view. For many tasks, such as pattern recognition, no single approach dominates the other. The choice of the best technique should be driven by the nature of the given application. We should try to understand the capacities, assumptions, and applicability of various approaches, and maximally exploit the complementary advantages of these approaches in order to develop better intelligent systems. Such an effort may lead to a synergistic approach which combines the strengths of ANNs and other approaches in order to achieve a significantly better performance for challenging problems. As Minsky [15] has observed, the time has come to build systems out of diverse components. In such a synergistic approach, not only are individual modules important, but we also need a good methodology for integration. It is clear that communication and cooperative work between researchers working in ANNs and other disciplines will not only avoid repetitious work but, more importantly, will stimulate and benefit individual disciplines.

Acknowledgment: The authors would like to thank Richard Casey (IBM Almaden), Pat Flynn (Washington State Univ.), William Punch, Chitra Dorai and Kalle Karu (Michigan State Univ.), Ali Khotanzad (Southern Methodist Univ.), and Ishwar Sethi (Wayne State Univ.) for their many useful suggestions.

References

- [1] *DARPA Neural Network Study*. AFCEA International Press, 1988.

- [2] James A. Anderson and Edward Rosenfeld. *Neurocomputing: Foundations of Research*. MIT Press, Cambridge, Massachusetts, 1988.
- [3] S. Brunak and B. Lautrup. *Neural Networks, Computers with Intuition*. World Scientific, Singapore, 1990.
- [4] G. A. Carpenter and S. Grossberg. *Pattern Recognition by Self-Organizing Neural Networks*. MIT Press, Cambridge, MA, 1991.
- [5] J. Feldman, M. A. Fanty, and N. H. Goddard. Computing with structured neural networks. *IEEE Computer*, pages 91–103, March 1988.
- [6] S. Haykin. *Neural Networks: A Comprehensive Foundation*. MacMillan College Publishing Company, New York, 1994.
- [7] D. O. Hebb. *The Organization of Behavior*. Wiley, New York, 1949.
- [8] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City, 1991.
- [9] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. In *Proc. Natl. Acad. Sci. USA 79*, pages 2554–2558, 1982.
- [10] A. K. Jain and J. Mao. Neural networks and pattern recognition. In J. M. Zurada, R. J. Marks II, and C. J. Robinson, editors, *Computational Intelligence: Imitating Life*, pages 194–212. IEEE Press, New York, 1994.
- [11] T. Kohonen. *Self Organization and Associative Memory*. Third edition, Springer-Verlag, 1989.
- [12] Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Back-propagation applied to handwritten zipcode recognition. *Neural Computation*, 1:541–551, 1989.
- [13] R. P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, 4 (2):4–22, Apr. 1987.

- [14] W. S. McCulloch and W. Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.
- [15] M. Minsky. Logical versus analogical or symbolic versus connectionist or neat versus scruffy. *AI Magazine*, 65(2):34–51, 1991.
- [16] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, 1969.
- [17] K. Mohiuddin and J. Mao. A comparative study of different classifiers for handprinted character recognition. In E. S. Gelsema and L. N. Kanal, editors, *Pattern Recognition in Practice IV*, pages 437–448. Elsevier Science, The Netherlands, 1994.
- [18] R. Rosenblatt. *Principles of Neurodynamics*. Spartan Books, New York, 1962.
- [19] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*. MIT Press, Cambridge, MA, 1986.
- [20] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Applied Mathematics, Harvard University, November 1974.
- [21] R. A. Wilkinson and J. Geist et al. (eds). The first census optical character recognition system conference. Technical report, NISTIR 4912, U.S. Department of Commerce, NIST, Gaithersburg, MD 20899, 1992.