Accepted to appear in IEEE Computer Special Issue on Neural Computing, March, 1996.

Artificial Neural Networks: A Tutorial

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

*Computer Science Department
Michigan State University
East Lansing, MI 48824

+IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120

Abstract

Numerous efforts have been made in developing "intelligent" programs based on the Von Neumann's centralized architecture. However, these efforts have not been very successful in building general-purpose intelligent systems. Inspired by biological neural networks, researchers in a number of 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 for several 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, basic network models, and two main issues in designing ANNs: network architecture and learning process. We also present one of the most successful application 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: (i) Pattern classification, (ii) Clustering/categorization, (iii) Function approximation, (iv) Prediction/forecasting, (v) Optimization, (vi) Retrieval by content, and (vii) Control. A number of successful attempts have been made to solve these problems using a variety of ANN models. Because of these successes, ANNs have now become a popular tool for problem solving.

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 prespecified classes. Discriminant functions or decision boundaries are constructed from a set of training patterns with known class labels to separate patterns from different classes. The decision boundaries can be linear, piece-wise linear, or any arbitrary shape (see Figure 1(a)). Two important issues in a pattern classification task are feature representation/extraction and decision making. 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 Figure 1(b)). The number of clusters is often not known *a priori*. Therefore, clustering is a more difficult problem than pattern classification. 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), $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_n, y_n)\}$, generated from an unknown function $\mu(\mathbf{x})$ (subject to noise), the task of function approximation is to find an estimate, say f, of the unknown function μ. In the statistical literature, this problem is often referred to as *regression*. The estimated function f can be made to fit the training data with an arbitrary accuracy by adjusting its complexity. An important issue here is to avoid *over-fitting* to the given noisy training data (see Figure 1(c)). Pattern classification can also be posed as a function approximation problem. Various engineering and scientific modeling problems require function

approximation.

Prediction/forecasting: Given a set of n samples in a time sequence, $\{y(t_1), y(t_2), \cdots, y(t_n) | t_1 < t_2 < \cdots < t_n\}$, 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, as well as our daily life. Stock market prediction and weather forecasting are typical applications of prediction/forecasting techniques (see Figure 1(d)).

Optimization: A wide variety of problems in mathematics, statistics, engineering, science, medicine, and economics can be posed as optimization problems. An optimization problem usually involves the following components: (i) a set of independent variables or parameters which is often referred to as the *state* of the process; (ii) an *objective* function or *cost/error* function to be optimized, and (iii) a set of constraints if they exist. The goal of an optimization algorithm is to find a state satisfying the constraints such that the objective function is maximized or minimized. A *combinatorial optimization* problem refers to a problem in which all the state variables are discrete and have a finite number of possible values. A classical combinatorial 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 does not have any physical meaning in terms 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 Figure 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 (Figure 1(g)). In this example, throttle angle is the control input, and engine speed is the output of the system. The reference

input (throttle angle) sets the engine at the desired idle speed, when the load torque is zero. Without an adaptive control system, various load torque values would set the engine at different idle speeds. The goal of the engine idle speed control system is to adaptively generate the throttle angle such that the engine runs at the desired idle speed at all load torques. Many other engineering systems require an adaptive control.

A large number of approaches have been proposed for solving the problems described in Figure 1. 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 that a large number of applications can benefit from the use of ANNs [1, 9, 7].

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. ANNs were designed with the goal of building "intelligent machines" to solve complex problems, such as pattern recognition and optimization, by mimicking the network of real neurons in the human brain (biological computation). Another goal of ANNs is to help us understand our brain through simulating and testing hypotheses about network architecture and learning.

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 a brief introduction to biological neurons and neural networks, motivations behind developing ANNs, relationship of ANNs to other scientific disciplines, and a brief historial note. In Section 3, we present the basic neuron model, and discuss the two main issues in designing ANNs: (i) network architecture, and (ii) learning process. Various ANN models are organized according to their architecture and learning process. Sections 4 - 7 provide more details about several well-known ANN models: Multilayer perceptron, Kohonen's Self-Organizing Maps, ART models and Hopfield network. In Section 8, we discuss character recognition, a popular domain for applying ANN models. Concluding remarks are presented in Section 9.

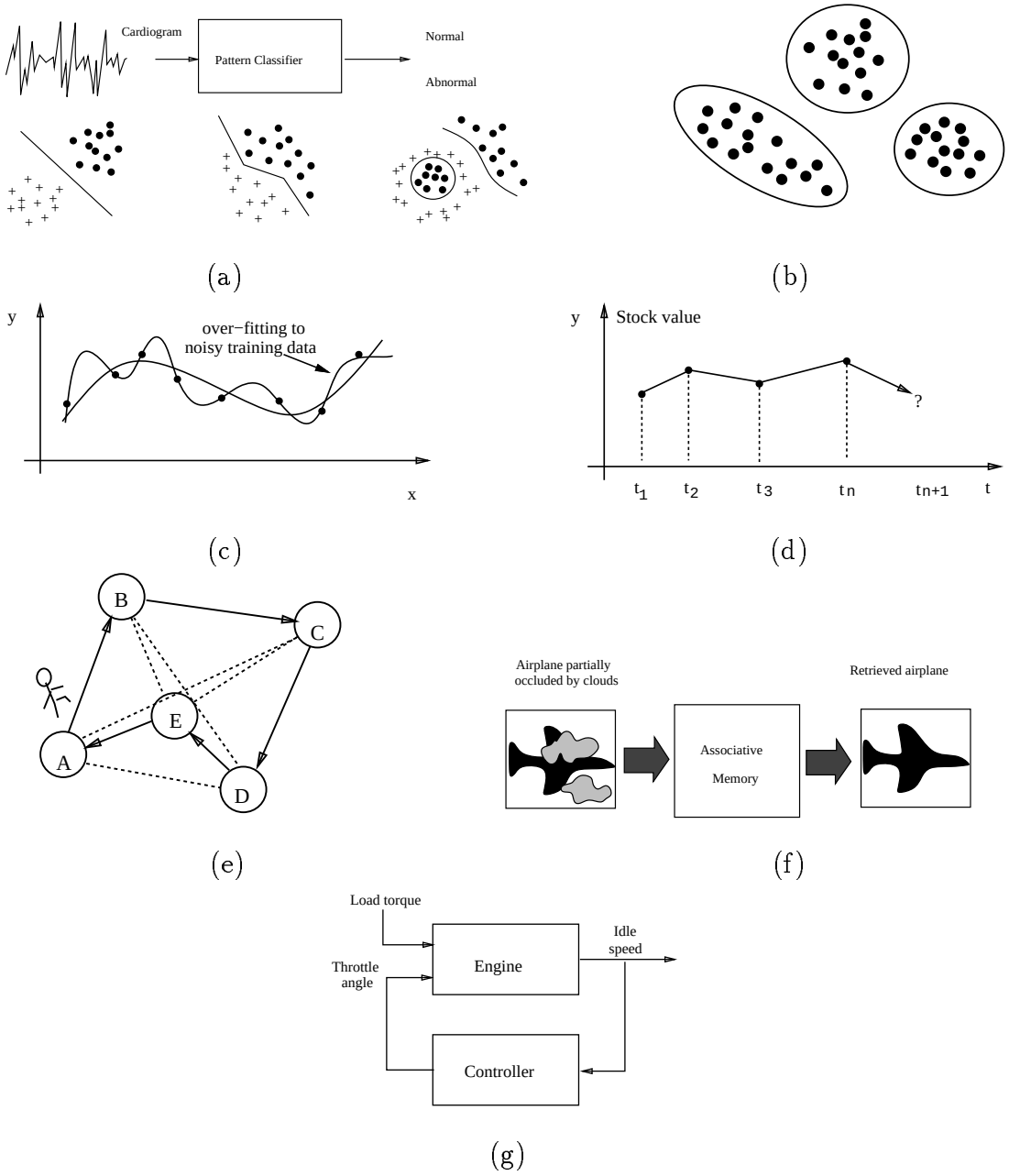


Figure 1: Tasks that neural networks can perform. (a) Pattern classification; (b) clustering/categorization; (c) Function approximation; (d) Prediction/forecasting; (e) Optimization (TSP problem); (f) Retrieval by content; and (g) Engine idle speed control.

2 Motivations

This tutorial article is about the fundamentals of artificial neural networks. However, we also need to provide a brief introduction to biological neural networks for the following reasons: (i) ANNs are inspired by biological neural networks; (ii) a network of massively connected simple processors (PDP model) exhibits powerful computational capabilities; (iii) the biological neural network provides a benchmark for evaluating the performance of ANNs; (iv) biological neural networks are an existence proof of our goal of building intelligent machines.

2.1 Biological neuron/neural networks

A *neuron* (or nerve cell) is a special biological cell, the essence of life, with information processing ability. The introduction of neurons as basic structural constituents of the brain was credited to Ramon y Cajal who won the 1906 Nobel prize for physiology and medicine (shared with Camillo Golgi) for the crucial discovery of the extensive interconnections within the *cerebral cortex*, the portion of the brain where approximately 90% of the neurons in the human are located.

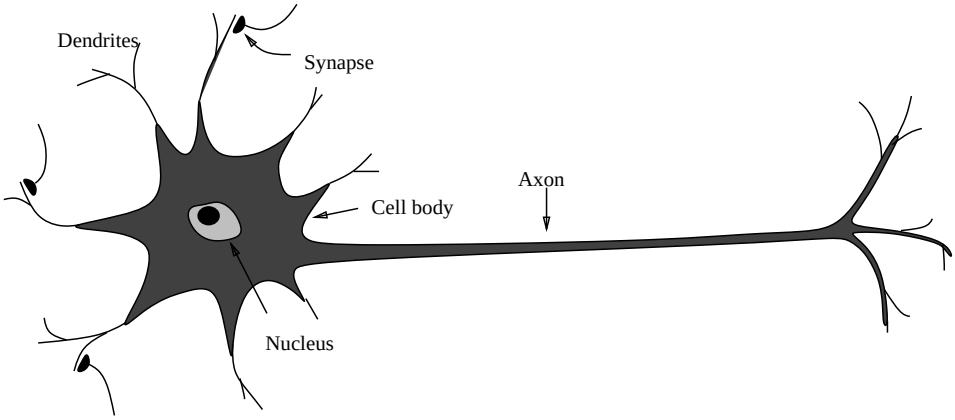


Figure 2: A sketch of a biological neuron.

A schematic drawing of a neuron is shown in Figure 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. The cell membrane contains various types of electrochemical pumps which can maintain imbalances in charge concentrations inside and outside the cell. 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 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. This is an amazing creation of nature because a sphere with a volume of about 1.5 liters, the typical size of a human brain, has a surface area of only 634 cm^2. It is the walnut appearance of human brain that provides the cerebral cortex with a surface area three times larger than a simple smooth spherical surface. The cerebral cortex contains about 10^{11} neurons, which is approximately the number of stars in the Milky Way! There are about 34 different types of neurons based solely on their shape, and as many as 100 types of functionally different neurons. 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. The number of interconnections depends on the location of the neuron in the brain and the type of the neuron. 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 at the brain very quickly, typically within a few hundred milliseconds. These decisions are made by a network of neurons whose operational speed is a few milliseconds. This implies that computation 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* [6]. 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, thus comes the name *connectionist* model. What is the magic that permits slow computing elements to perform extremely complex tasks rapidly? The key is the parallel and distributed representation and computation.

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 fast 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.

Numerous efforts have been made on developing "intelligent" programs based on the Von Neumann's centralized architecture. However, such efforts have not resulted in any general-purpose intelligent programs. ANNs 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. This, of course, implies that the achievement of ANNs is largely dependent on the depth of our understanding of the human brain, which is beyond our comprehension. On

	Von Neumann computer	Biological computer
Processor	complex	simple
	high speed	low speed
	one or a few	large number
Memory	separate from 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.

the other hand, a successful ANN may not have any resemblance to the biological system. Our ability to model biological nervous system using ANNs can increase our understanding of biological functions. For example, experimental psychologists have used neural networks to model classical conditioning animal learning data for many years [1]. The state-of-the-art in computer hardware technology (e.g., VLSI and optical) has made such modeling and simulation feasible.

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

istics in the human brain.

2.3 Relationship with other disciplines

The field of artificial neural networks is an interdisciplinary area of research. A thorough study of artificial neural networks requires 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).

Figure 3 illustrates the interaction between artificial neural networks and these disciplines. Artificial neural networks receive inputs from these disciplines. 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 terms of new tools and representations. This symbiosis is necessary for the vitality of neural network research. Communications among these disciplines ought to be encouraged.

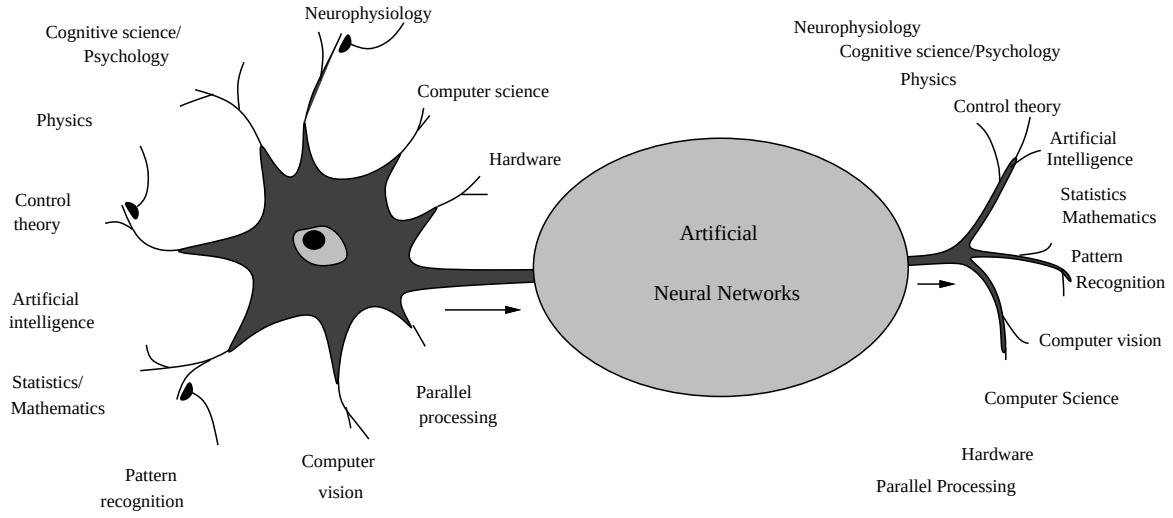


Figure 3: ANN's Relationship with other disciplines.

We shall not discuss all these interactions between ANNs and other disciplines. Instead, we shall primarily focus on the relationship between ANNs and pattern recognition and artificial intelligence.

Pattern recognition systems are expected to automatically classify, describe, or cluster complex patterns or objects based on their measured properties or features. Design of a pattern recognition system involves the following three main steps: (i) data acquisition and preprocessing, (ii) representation or feature extraction, and (iii) decision making or clustering. Jain and Mao [11] have addressed a number of common links between ANNs and statistical pattern recognition (SPR). There is a close correspondence between some of the popular ANN models and traditional pattern recognition approaches. Quite often, these relationships are not fully exploited to build hybrid systems. Examples are perceptron versus linear classifier, vector quantization learning by ANNs versus k-means clustering, and radial basis function network versus Parzen window density estimation/classifier. In spite of this close resemblance between ANN and SPR, ANNs have provided a variety of novel or supplementary approaches for pattern recognition tasks. More noticeably, ANNs have provided architectures on which many well-known statistical pattern recognition algorithms can be mapped to facilitate hardware implementation. The adaptivity of ANNs is crucial for the design of pattern recognition systems not only in terms of good generalization capability, but also in terms of its performance in dynamic environments and in the presence of incomplete information during training. At the same time, ANNs can derive benefit from some well-known results in statistical pattern recognition. For example, the generalization ability of a network is related to the "curse of dimensionality" problem in statistical classifiers; radial basis function networks share many design issues with Parzen window classifiers. Most of the efforts in the ANN research related to pattern recognition have been directed to designing a classifier with good generalization ability. Little work has been devoted towards designing a good representation scheme for a given problem using neural networks.

Artificial intelligence (AI) aims at building "intelligent" machines that can perform tasks which require *cognition* when performed by humans. A typical AI system has three major components: representation, reasoning, and learning (see [7]). With motivation derived from psychology and cognitive science (mental representation), natural language (symbols, sequential processing), and logic (symbol manipulation), traditional AI systems adopt (i) *symbolic representation*, (ii) *searching-based reasoning* using rules, logic, and knowledge database, and (iii) expert-based learning (expert systems). AI takes the *top-down* strategy to solve prob-

lems [16]: begin at the level of commonsense psychology, and hypothesize what processes could solve a problem. If the problem can not be solved in a single step, break the problem into subproblems. This procedure continues until a solution is obtained.

In contrast, ANNs distinct themselves from traditional AI by employing the parallel and distributed processing (PDP) model which uses a network of massively connected simple processing units (connectionist model). Problems, knowledge, and even solutions are represented (coded) by numeric weights and outputs which are distributed in the network. Motivated by neurophysiology, ANNs take the *bottom-up* strategy: start from simple processing units, and then move upward in complexity by studying their interconnections and collective behavior.

Both AI and ANN paradigms have their own virtues and deficiencies (symbolic versus connectionist, top-down versus bottom-up) [16]. Most importantly, the virtues of one approach could compensate the deficiencies of the other. Therefore, we should not exclude any one of these two approaches based on our bias. Instead, a useful approach might be to combine both the approaches in building *structured connectionist models*.

2.4 Brief Historical Review

Humans, being inquisitive creatures, have long been interested in exploring where the mind originates and how the brain computes. These efforts may be traced back to Aristotle. Yet, the modern era of computational neural modeling began with the pioneering work of McCulloch and Pitts [15] in 1943, who introduced a computational model of neuron and a logical calculus of neural networks. McCulloch-Pitts' classic paper was widely read at the time (and is still read), generating considerable interest over the next 15 years in the detailed logic of networks consisting of their simple neurons. Such networks were proved to be capable of *universal computation* (any Boolean function).

The next major milestone in ANNs was Rosenblatt's work on the Perceptron in 1958. The crowning achievement of Rosenblatt's work was the first proof of the perceptron convergence theorem. In 1960, Widrow and Hoff introduced the least mean square (LMS) algorithm for the Adaline (Adaptive Linear Element). Nilsson's book on machine learning [19] was the best-written exposition of linearly separable patterns in hypersurfaces. ANNs generated a

great deal of enthusiasm in the 1960's. It appeared as if such a machine could do any type of computation. However, this enthusiasm was dampened by Minsky and Papert's book [17] which demonstrated the fundamental limitations of the computing power of one-layer perceptrons. They showed that certain rather simple computations, such as the *Exclusive-OR* (XOR) problem, could not be solved by the one-layer perceptron. It was believed that such limitations could be overcome by multilayer perceptrons which employ intermediate layers of units (hidden units) between the input layer and output layer. But, a difficult problem encountered in designing a multilayer perceptron is the credit assignment problem (i.e., the problem of assigning credit to the hidden units in the network). There was no learning algorithm known at that time to solve this problem. Minsky and Papert doubted that one could be found and thought it more profitable to explore other approaches to artificial intelligence. Because of this and other reasons, research into neural networks went into hibernation. However, the neural network field was not completely abandoned in the 1970's. A number of dedicated researchers continued to develop neural network models. Two important themes that emerged were associative content-addressable memory and self-organizing networks using competitive learning.

In the 1980's, a number of important publications appeared, which changed the course of ANN research. Perhaps more than any other publication, the 1982 paper by Hopfield [10] and the two-volume book by Rumelhart and McClelland in 1986 [21] were the most influential publications. In 1982, Hopfield introduced the idea of an energy function from statistical physics to formulate a new way of understanding the computation of recurrent networks with symmetric synaptic connections. This formulation makes explicit the principle of storing information as dynamically stable attractors. Many combinatorial optimization problems, such as the classical Traveling Salesperson Problem, can be formulated in terms of a network energy function which is minimized when the network reaches a stable state.

In 1986 Rumelhart, Hinton and Williams reported the development of the backpropagation algorithm which popularized the use of multilayer perceptron to solve a wide variety of pattern recognition problems. In fact, the development of the back-propagation algorithm has a colorful history. It was first developed by Werbos in 1974 in his Ph.D. thesis, and later rediscovered independently in two other places by Parker and by LeCun in 1985.

Over the last ten years, thousands of researchers from many diverse fields, such as neuroscience, psychology, medicine, mathematics, physics, computer science, and engineering, have been involved in developing neural network models, implementing the models in hardware (VLSI and optics) and software, and solving a number of important applications. These activities continue to grow as a result of the successful applications of the ANN models.

3 Artificial Neurons/Neural Networks

This section provides an overview of ANNs. First, computational models of neurons are introduced. Then, two important issues, 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 [15] proposed a binary threshold unit as a computational model for a neuron. A schematic diagram of a McCulloch-Pitts neuron is shown in Figure 4. This

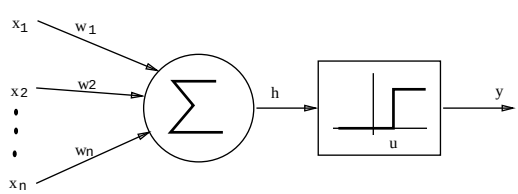


Figure 4: McCulloch-Pitts model of a neuron.

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

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

where $\theta(\cdot)$ is a unit step function, and w_j is the synapse weight associated with the jth input. For simplicity in notation, we often consider the threshold μ as another weight $w_0 = -\mu$

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 (Table 2) 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. Some of these differences are:

- Biological neurons are not threshold devices, but have graded response (essentially a nonlinear function of the inputs);
- Biological neurons perform a nonlinear summation of inputs and can even perform logical processing;
- Biological neurons produce a sequence of pulses, not a simple output value;
- Biological neurons are updated asynchronously.

Nevertheless, the McCulloch-Pitts neuron model started a new era for computational neural modeling.

Biological neurons	Artificial neurons
Synapses	Connection weights
Axons	Output wires
Dendrites	Input wires
Soma	Activation function

Table 2: An analogy between biological neurons and artificial 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 Figure 5. The sigmoid function is by far the most frequently

used function in ANNs. It is a strictly increasing function that exhibits smoothness and 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.

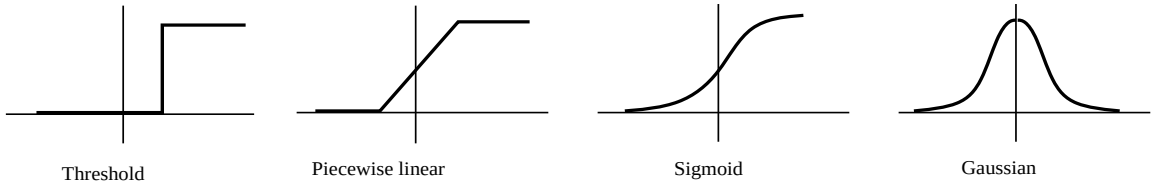


Figure 5: Different types of activation functions.

3.2 Network Architecture/Topology

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

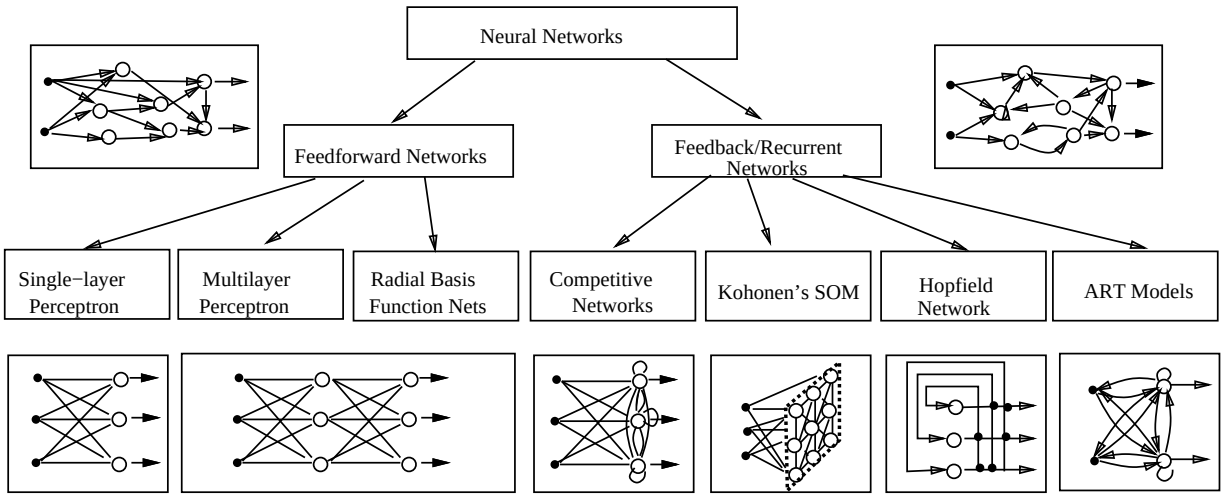


Figure 6: A taxonomy of network architectures.

on the connection pattern (architecture), various ANNs can be grouped into two major

categories as shown in Figure 6: (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. In fact, all the networks with no loops can be rearranged in the form of layered feedforward networks with possible skip-layer connections. Figure 6 also shows typical networks of each category. We will discuss in this article all these networks except the Radial Basis Function (RBF) networks (see [7]) which employ the same network architecture as multilayer perceptrons, but different activation functions.

Different connectivities exhibit different network behaviors. Generally speaking, feedforward networks are *static* networks, i.e., given an input, they produce only one set of output 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. An exception is the time delay feedforward network in which dynamics occurs because of different delay factors of the neurons in 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. This process is repeated until convergence. Obviously, different mathematical tools must be employed to treat these two different types of networks. Dynamic systems are often described by differential equations.

These network architectures can be either simulated in software or implemented in hardware (VLSI and optical). Software simulation of a network is always necessary before implementing it in hardware. A number of public and commercial software ANN simulators are available. More and more researchers have recognized the importance of hardware implementation, which is probably the only way to take the full advantage of the capacities of ANNs. A difficulty in the VLSI implementation of ANNs is the massive connections. A fully connected network with N neurons requires N^2 connections! This factor limits the number of neurons (typically a few hundred) that we can build on a single chip using the state-of-the-art VLSI technology. An alternative is the optical implementation of ANNs.

But, it is still in the early stages.

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

3.3 Learning

Ability to learn is a fundamental trait of intelligence. Although what is meant by learning is often difficult to describe, 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. Typically, learning in ANNs is performed in two ways. Sometimes, weights can be set *a priori* by the network designer through a proper formulation of the problem. However, most of the time, the network must learn the connection weights from the given training patterns. Improvement in performance is achieved over time through iteratively updating the weights in the network. The ability of neural networks to automatically *learn from examples* makes artificial neural networks very attractive and exciting. Instead of having to specify a set of *rules*, ANNs appear to learn 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 [7]. 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. Finally, it is important to investigate how much the network can learn from examples (capacity), how many training samples are required (sample complexity), and how fast the system can learn (time complexity). The study of capacity, sample complexity, and time complexity is what a *learning theory* must deal with. Figure 7 illustrates these three aspects of a learning process.

There are three main learning paradigms, namely, (i) supervised, (ii) unsupervised, and (iii) hybrid learning. In *supervised learning*, the network is provided with a correct answer to every input pattern. Weights are determined so that the network can produce answers as

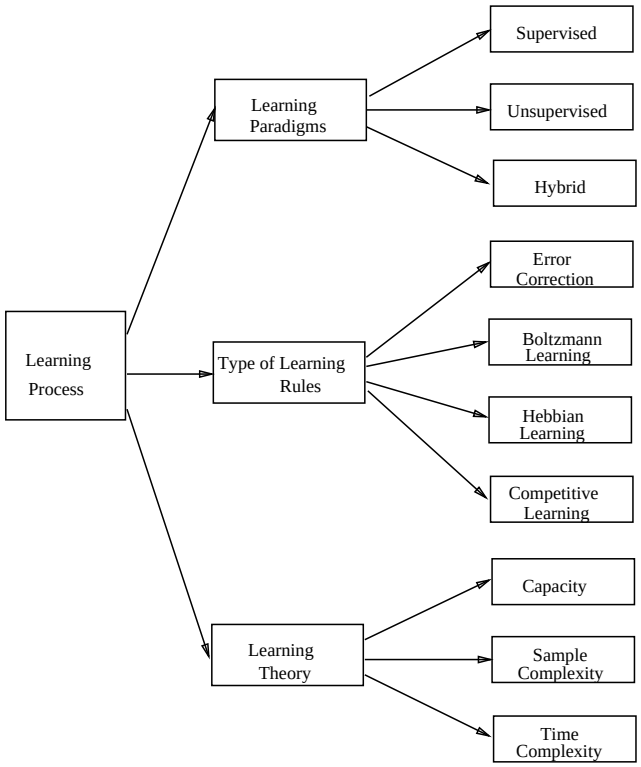


Figure 7: Learning issues.

close as possible to the known correct answers. This is sometimes referred to as *learning with a teacher*. *Reinforcement* learning is a special case of supervised learning where the network is provided with only critiques on the correctness of network outputs, not the correct answers (outputs) themselves. In contrast, *unsupervised learning* 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. The first issue concerns whether the true solution is contained in the set of solutions that a network can deliver. If not, we can never hope to obtain the optimal solution. This remains

a difficult and open problem. The approximation capabilities of feedforward neural networks have recently been investigated by many researchers (see, [9]). A fundamental result of these studies is that 3-layer, or even 2-layer, feedforward networks with an arbitrarily large number of nonlinear hidden units are capable of implementing any continuous mapping with a pre-specified accuracy under certain mild conditions. Unfortunately, most of these theoretical studies ignore the learnability problem that is concerned with whether there exist methods to learn the network weights from empirical observations of the mappings. Furthermore, these theoretical analyses have not introduced any new practical learning methods.

The second issue, 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 the "over-fitting" problem where the network performs well on the training data set, but poorly on independent test patterns drawn from the same distribution as the training patterns.

The third issue is the computational complexity of the learning algorithm used to estimate a solution from the training patterns. Many existing learning algorithms have high computational complexity. For example, the popular backpropagation learning algorithm for feedforward networks is computationally demanding because of its slow convergence. Designing efficient algorithms for neural network learning is a very active research topic.

There are four basic types of learning rules as shown in Figure 7: (i) error-correction, (ii) Boltzmann, (iii) Hebbian, and (iv) competitive learning. They will be described in the following four subsections.

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 the error-correction principle. A perceptron consists of a single neuron with adjustable weights, w_j, $j = 1, 2, \cdots, n$, and

threshold μ, as shown in Figure 4. Given an input vector $\mathbf{x} = (x_1, x_2, \cdots, x_n)^t$, 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 [20] 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 $(x_1, x_2, \cdots, x_n)^t$, 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.* However, 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 [9]. 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 monotonous activation function is used.* Note that non-monotonous activation functions, such as a Gaussian function, could form non-linear decision boundaries.

Figure 8 shows the trajectory of the decision boundary learned using a modified perceptron learning algorithm for classifying the logical "AND" problem which is linearly-separable.

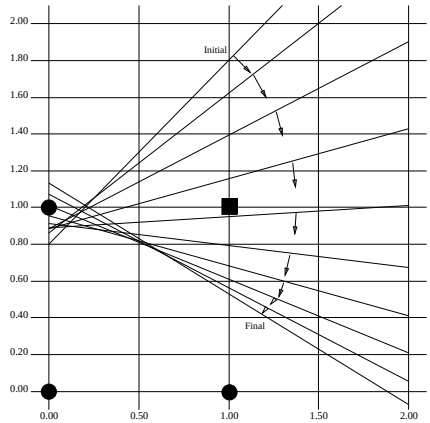


Figure 8: Convergence of a modified perceptron learning algorithm for the logical "AND" problem.

The well-known backpropagation learning algorithm (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 (hidden neurons) are invisible. Each neuron is a stochastic unit which generates 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 of connection weight w_{ij} is given by

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

where $\overline{\rho}_{ij}$ 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 $\overline{\rho}_{ij}$ 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 rule 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, yet still used, learning rule is *Hebb's postulate of learning* [8]. It was proposed by Hebb based on the following observation from neurobiological experiments: *When an axon of a cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic changes take place in one or both cells such that A's efficiency as one of the cells firing B is increased.* In other words, if two neurons on the either side of a synapse are activated synchronously and repeatedly, then the strength of that synapse is selectively increased [7]. Such a synapse is often referred to as *Hebb's synapse*, or *correlational synapse* because the change of the synapse's strength depends on the correlation between the presynaptic and postsynaptic activities.

Mathematically, the update of Hebb's synapse 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.

One problem with this learning rule is that the connection weights will grow unboundedly as learning proceeds. To deal with this problem, many modifications to the basic Hebbian rule have been proposed [7, 9]. For example, Oja's rule adds a weight decay proportional to y_j^2 to the basic Hebbian rule:

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

It is interesting to note that this rule is similar to the reverse error-correction rule; Δw_{ij} depends on the difference between the actual input and the back-propagated output.

A single neuron trained using the Hebbian rule exhibits an orientation selectivity. Figure 9 demonstrates this property. The points depicted in Figure 9 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.

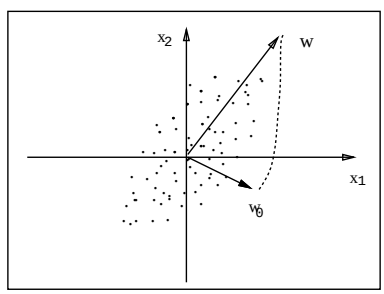


Figure 9: Orientation selectivity of a single neuron.

It is straightforward to generalize the above behavior of a single unit and conclude that a one-layer feedforward network with m output units can extract first m principal components of n-dimensional data, $m \leq n$. Due to the orthogonality of the eigenvectors, the other $(m-1)$ principal components lie in the subspace which is perpendicular to the first principal component corresponding to the largest eigenvalue. Therefore, the $(m-1)$ principal components can be determined recursively in subspaces in a way similar to computing the first component. Several more elegant methods have been proposed for computing all the

principal components simultaneously by imposing some constraints on the activities of the output units [7].

3.3.4 Competitive Learning Rules

Unlike the Hebbian learning where multiple output units can be fired simultaneously, in competitive learning all the output units compete among themselves for being activated. As a result of such competition, only one output unit, or only one per group, 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. Neurobiological experiments have shown that competitive learning plays an important role in the formation of topographic maps in the brain, and the self-organization of orientation sensitive nerve cells in the striate cortex.

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 Figure 10. Each output unit i in the network connects to all the input units

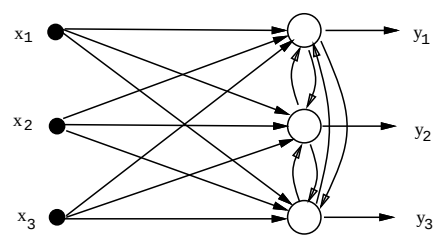


Figure 10: A simple competitive learning architecture.

via weights, w_{ij}, $j = 1, 2, \cdots, d$. Each output unit also connects to all the other output units via inhibitory weights, but has self-feedback with an excitatory weight. As a result of competition, only the unit with the largest (or the smallest) net input becomes the winner, i.e.,

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

or

$$\|\mathbf{w}_{i^*} - \mathbf{x}\| \leq \|\mathbf{w}_i - \mathbf{x}\| \quad \forall\, i.$$

When all the weight vectors are normalized, these two inequalities are equivalent.

A simple competitive learning rule can be stated as follows.

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

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 Figure 11. In this example, we assume that all the input vectors have been normalized to have unit length. They are depicted as black dots in Figure 11(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 Figures 11(a) and 11(b), respectively. As we can see from Figure 11, each of the three natural groups of patterns has been discovered by an output unit whose weight vector points to the center of gravity of the discovered group.

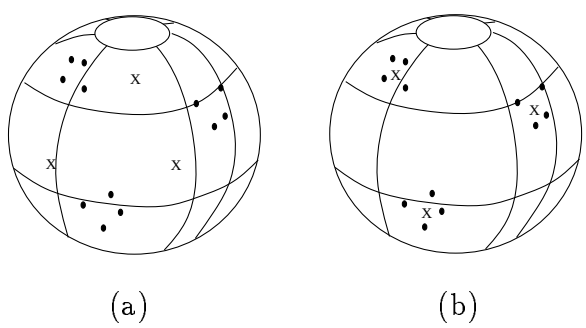


Figure 11: An example of competitive learning: (a) before learning; (b) after learning.

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 pattern may fire different output units (change categories) forever during the 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 of *plasticity*, which is defined as the ability to adapt to new data. This is the well-known Grossberg's *stability-plasticity* dilemma in competitive learning.

Perhaps, 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 can 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. If the Euclidean distance is used, this divides the input space into a *Voronoi tessellation*. The competitive learning rule in Equation (1) can be used for generating a codebook for a given set of input vectors.

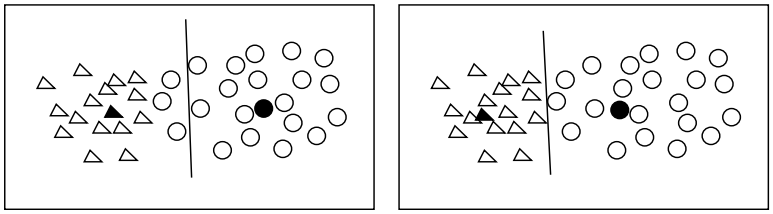
The codebook and Voronoi tessellation generated by the unsupervised competitive learning rule may not be the best for pattern classification purposes (see Figure 12(a)). *Learning vector quantization* (LVQ) [12] is a supervised competitive learning technique which uses pattern class information to adjust the Voronoi vectors slightly, so as to improve classification accuracy. In LVQ, the weight updating rule is replaced by

$$w_c(t+1) = \begin{cases} w_c(t) + \eta(t)[x(t) - w_c(t)], & \text{If pattern } x(t) \text{ is correctly classified} \\ & \text{by the winning unit } c, \\ w_c(t) - \eta(t)[x(t) - w_c(t)], & \text{otherwise.} \end{cases}$$

Figure 12(b) demonstrates the effect of LVQ. It moves the prototypes learned using the VQ algorithm slightly to the left in order to classify all the patterns correctly.

3.3.5 Summary of Learning Algorithms

Various learning algorithms and their associated network architectures are summarized in Table 3. However, this is by no means an exhaustive list of the learning algorithms available in the literature. We notice that both the supervised and unsupervised learning paradigms



(a) (b)

Figure 12: Vector quantization (a) versus learning vector quantization (b). Patterns from two classes are labeled by triangles and circles, respectively. Solid patterns are learned prototypes.

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 also designed for performing one or a few specific tasks. The last column of table 3 lists a number of tasks that each learning algorithm can perform. Due to space limitation, we will not discuss some of the other algorithms, including ADALINE, MADALINE [22], linear discriminant analysis (see [11]), ART2, ARTMAP [4], Sammon's projection (see [11]), principal component analysis (see [9]), and RBF learning algorithm (see [7]). Interested readers can further read the corresponding references. Note that in order to reduce the size of the bibliography, this article does not always cite the first paper that proposed a particular algorithm.

Learning 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 control
	Boltzmann	Recurrent	Boltzmann Learning algorithm	pattern classification
	Hebbian	Multi-layer Feedforward	Linear Discriminant Analysis	data analysis pattern classification
	Competitive	Competitive	Learning Vector Quantization	within-class categorization data compression
		ART network	ARTMAP	pattern classification within-class categorization
Unsupervised	Error-correction	Multi-layer Feedforward	Sammon's projection	data analysis
	Hebbian	Feedforward or Competitive	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 and Competitive	RBF network	RBF Learning algorithm	pattern classification function approximation control

Table 3: Well-known learning algorithms.

4 Multilayer Perceptron

It has been recognized that multilayer feedforward networks are capable of forming arbitrarily complex decision boundaries and can represent any Boolean function [17]. The development of the *back-propagation* learning algorithm for determining weights in a multi-layer feedforward network has made these networks the most popular of all the networks.

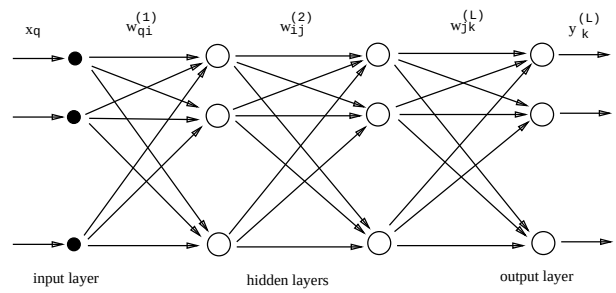


Figure 13: A typical 3-layer feedforward network architecture.

Figure 13 shows a typical 3-layer perceptron. In general, a standard L-layer feedforward network[1] 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. We denote ${w_{ij}}^{(l)}$ as the weight on connection between the ith unit in layer $(l-1)$ to j^{th} unit in layer l.

Recall that the task of a learning algorithm is to automatically determine the weights in the network such that a certain cost function is minimized.

Let $\{(\mathbf{x}^{(1)}, \mathbf{d}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{d}^{(2)}), \cdots, (\mathbf{x}^{(p)}, \mathbf{d}^{(p)})\}$ 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 $\mathbf{d}^{(i)} \in [0,1]^m$ is the desired output vector in the m-dimensional hyper-cube. For classification purposes, m is set to the number of classes. The squared-error cost function, which is most frequently used in the ANN literature, can be defined as

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

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

The back-propagation algorithm is a gradient-descent method to minimize the above squared-error cost function in Equation (2). It can be described as follows [21].

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 ith unit in the lth layer.
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 $l = (L-1), \cdots, 1$.
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 the maximum number of iterations is reached.

A geometric interpretation (adopted and modified from [14]) shown in Figure 14 can help us understand 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. Arrange the output-layer units to perform an "OR" operation on all the second-layer units. 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. Moreover, multilayer feedforward networks with sigmoid activation functions can form smooth decision boundaries rather than piece-wise linear boundaries.

There are many issues in designing feedforward networks. These issues include: (i) how

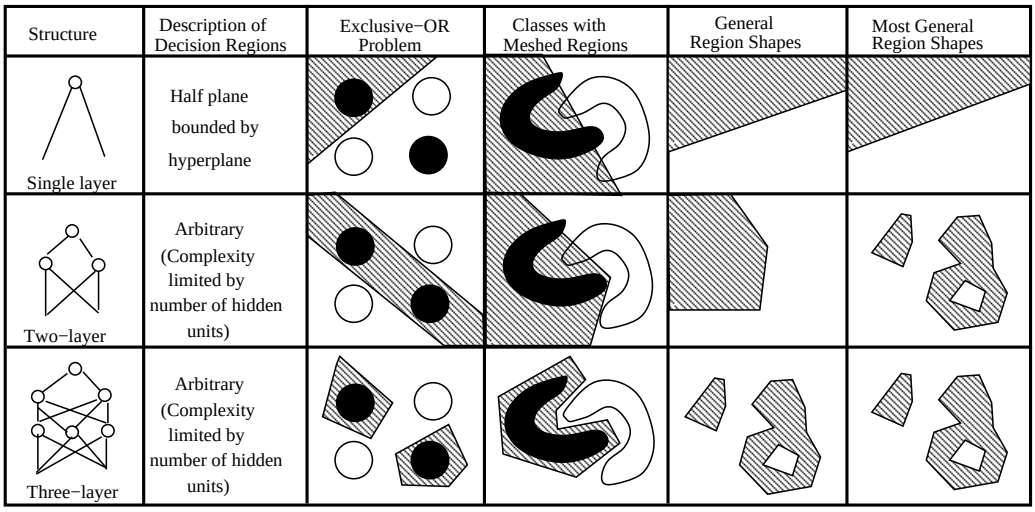


Figure 14: A geometric interpretation of the role of hidden units.

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 with backpropagation algorithm has been widely used for classification and function approximation (see [9]), many design parameters still have to be determined by the trial-and-error method. Existing theoretical results only provide very loose guidelines for selecting these parameters in practice.

5 Kohonen's Self-Organizing Maps

Kohonen's Self-Organizing Map (SOM) [12] has the desirable property of topology preserving which captures an important aspect of the feature maps in the cortex of the more developed animal brains. By 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 Figure 15. It basically consists of a two-dimensional array of units, each of which is connected to all the 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

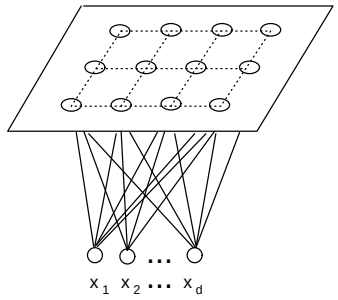


Figure 15: Kohonen's Self-Organizing Map.

between the input vector $\mathbf{x}$ and the stored weight vector $\mathbf{w}_{ij}$.

$$y_{ij} = ||\mathbf{x} - \mathbf{w}_{ij}||.$$

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, rectangle, or circle. Initial neighborhood size is often set to 1/2 to 2/3 of the network size. Neighborhood shrinks with time according to some schedule such as an exponentially decreasing function. During the 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 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 the 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 [9]. 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 learning rate.

6 Adaptive Resonance Theory Models

Recall that an important issue in competitive learning is the *stability-plasticity* dilemma. How can our brain learn new things (plasticity) yet retain the stability that ensures the existing knowledge not being 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 (not) being used. The learning algorithm updates the stored prototypes of a category only if the input vector is sufficiently similar to them.

Input and a stored prototype are said to resonate when they are sufficiently similar. The sufficient 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.

ART1 takes only binary (0/1) input, while ART2 was designed for continuous-valued input. The newer version ARTMAP makes use of pattern label information. Here, we present only ART1 to illustrate the model.

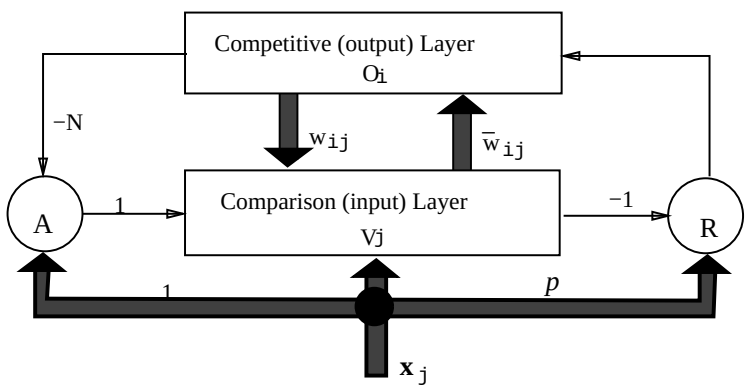


Figure 16: ART1 network.

Figure 16 shows a simplified diagram of the ART1 architecture (see [9]). It consists of two layers of units, which are fully connected. Top-down weight vector $\mathbf{w}_j$ is associated with unit j in the input layer, and bottom-up weight vector $\overline{\mathbf{w}}_i$ is associated with output unit i; $\overline{\mathbf{w}}_i$ is the normalized version of $\mathbf{w}_i$.

$$\overline{\mathbf{w}}_i = \frac{\mathbf{w}_i}{\varepsilon + \sum_j w_{ji}}, \tag{3}$$

where ε is a small number which is used for breaking the ties in selecting the winner. Given an N-bit input vector $\mathbf{x}$, the output of the auxiliary unit A is given by

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

and the outputs of input units is given by

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

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 all the enabled output units
$$\overline{\mathbf{w}}_{i^*} \cdot \mathbf{x} \geq \overline{\mathbf{w}}_i \cdot \mathbf{x}, \ \forall i$$
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 runs entirely autonomously. It is able to create new categories and to reject an input pattern when the network reaches its capacity. However, the number of categories in the input data discovered by ART1 is sensitive to the vigilance parameter.

7 Hopfield network

The Hopfield network is a special type of recurrent network which uses the network *energy* function as a tool for designing recurrent networks and for understanding its dynamic behavior [10]. It is Hopfield's formulation that 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 ith 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

$$v_i = Sgn(\sum_j w_{ij} v_j - \theta_i), \tag{4}$$

where $Sgn(x)$ is the *signum* function which produces $+1$ if $x \geq 0$ and -1, otherwise. The network dynamics for the continuous Hopfield network is

$$\tau_i \frac{du_i}{dt} = -u_i + \sum_j w_{ij} g(u_j) - R_i \theta_i, \tag{5}$$

where u_i is the net input (potential) to the ith unit, g is the sigmoid function, $v_i = g(u_i)$, and R_i and τ_i are constants. At an equilibrium point,

$$v_i = g(\sum_j w_{ij} v_j - \theta_i). \tag{6}$$

The dynamic update of network states in Equation (4) can be carried out in at least two ways: *synchronously* versus *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 asynchronous updating scheme is more natural for biological networks. For the continuous Hopfield network, in addition to the synchronous and asynchronous updating schemes, Equation (5) provides a *continuous* updating scheme which is particularly desirable for circuit implementation.

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

$$E = -\frac{1}{2} \sum_i \sum_j w_{ij} v_i v_j. \tag{7}$$

The network energy of the continuous Hopfield network is defined as

$$E = -\frac{1}{2}\sum_i \sum_j w_{ij} v_i v_j + \sum_i \frac{1}{R_i}\int_0^{v_i} g^{-1}(x)dx + \sum_i \theta_i v_i. \tag{8}$$

The central property of these energy functions is that as the state of network evolves according to the network dynamics (Eqs. (4) and (5)), the network energy always decreases, and eventually reaches a local minimum point where the network stays with a constant energy. Such local minimum points in the state space are often referred to as *attractors*, due to the fact that starting with any point (or state) in the neighborhood of an attractor, the network will evolve into this attractor. Such a neighborhood is called the basin of attraction of an attractor. A sequence of state changes is named a *trajectory*. Figure 17 schematically explains these terminologies.

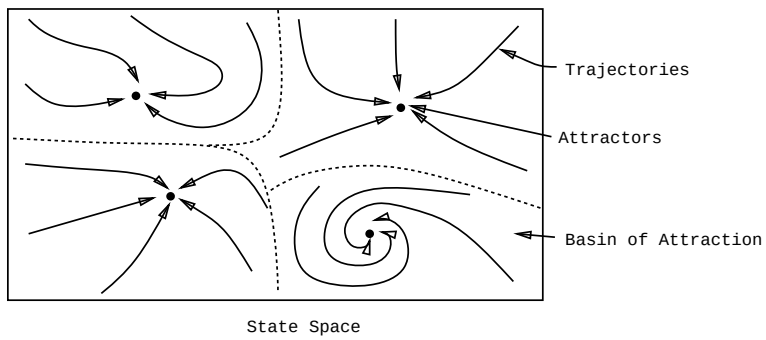


Figure 17: A schematic plot of attractors, trajectories, and basin of attraction.

Suppose a set of patterns are stored in these attractors of a network. Then this network can be used as an *associative memory*. A stored pattern can be retrieved by any pattern (represented by a network state) in the basin of attraction of the attractor corresponding to the stored pattern. This is the principle of using Hopfield network as an associative memory.

The attractors of a network can also encode the solutions of a combinatorial optimization problem if its cost (or objective) function can be formulated as the network energy. An optimal or sub-optimal solution is obtained as the network evolves into a local minimum point. This is the basic idea behind the use of Hopfield network for solving combinatorial optimization problems.

The next two subsections will discuss these two applications of Hopfield network.

7.1 Associative Memory

A fundamental property of associative or content-addressable memory is its ability to store a set of patterns in such a way that when presented with a new pattern which could be an incomplete or noisy version of a stored pattern, the network can retrieve one of the stored patterns which most closely resembles the input pattern. This property has a two-fold meaning. First, the memory must be accessed by content. Second, the memory must be error-correcting, i.e., an item in the memory can be reliably retrieved by noisy or incomplete information, as long as the information is sufficient.

Associative memory usually operates in two phases: storage and retrieval. In the storage phase, the weights in the network are determined or learned in such a way that the attractors of the network memorize a set of p N-dimensional patterns $\{\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^p\}$ to be stored. A generalization of Hebbian learning rule can be used for setting connection weights w_{ij}. Note that the number of units in the network equals to N.

$$w_{ij} = \begin{cases} \frac{1}{N}\sum_{\mu=1}^{p} x_i^\mu x_j^\mu, & i \neq j \\ 0 & i = j. \end{cases}$$

The values of all the thresholds θ_i are set to zero.

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 the states 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 [9].

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 almost a 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. The hardware efficiency of Hopfield network is extremely low, because we require N^2 connections in the network to store p N-bit patterns. Several learning rules have been proposed for increasing the memory capacity of Hopfield networks (see [9]).

7.2 Combinatorial optimization

Hopfield networks always evolve in the direction that leads to 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. We present here classical Traveling Salesperson Problem as an example of how a network is constructed.

The units in the network are organized into a two-dimensional $n \times n$ array, where n is the total number of cities. Let the row index of a unit represent a city, and the column index be the index of the stop in a tour. Let the output of the unit in row X and column i be v_{Xi}; $v_{Xi} = 1$ means the city X is visited at the ith stop. Therefore, a solution to the TSP problem is presented in an $n \times n$ permutation matrix of v_{Xi}'s. Let d_{XY} be the distance between city X and city Y. Now, we can construct the cost function for the TSP problem as follows.

We want to minimize the total distance

$$E_1 = \frac{D}{2} \sum_X \sum_{Y \neq X} \sum_i d_{XY} v_{X,i} (v_{Y,i+1} + v_{Y,i-1}).$$

We employ a periodic boundary condition, i.e., the $(n+1)^{th}$ column and the 0th column are the same as the first column and the last column, respectively.

The problem constraints are as follows:

(i) Each city must be visited once

$$E_2 = \frac{A}{2} \sum_X \sum_i \sum_{j \neq i} v_{X,i} v_{X,j}$$

(ii) Each stop must contain one city

$$E_3 = \frac{B}{2} \sum_i \sum_X \sum_{Y \neq X} v_{X,i} v_{Y,i}$$

(iii) The matrix must contain n entries

$$E_4 = \frac{C}{2}(\sum_X \sum_i v_{X,i} - n)^2.$$

The positive constants A, B, C, and D are the parameters of the problem. The total cost of the tour is defined as

$$E = E_1 + E_2 + E_3 + E_4. \tag{9}$$

Note that Equation (9) is quadratic in network outputs. After manipulating Equation (9), we obtain a quadratic function with three terms. The coefficients of the quadratic terms in the cost function define the connection weights in the network

$$w_{Xi,Yj} = -A\delta_{XY}(1-\delta_{ij}) - B\delta_{ij}(1-\delta_{XY}) - C - Dd_{XY}(\delta_{j,i+1} + \delta_{j,i-1}), \tag{10}$$

where

$$\delta_{ij} = \begin{cases} 1 & \text{if} \quad i = j \\ 0 & \text{if} \quad i \neq j \end{cases}$$

The coefficients of the linear terms specify the thresholds of the units.

$$\theta_{Xi} = -nC. \tag{11}$$

The constant term in the total cost does not change the solution and, therefore, it can be ignored.

A solution to the TSP problem, given a set of cities and distances, can thus be found either by running a physical network whose weights and thresholds are determined by Equations (10) and (11), or by computer simulation. It has been found that the continuous network performs better than the binary network. The latter easily gets stuck in a local minimum with poor tour length. *Simulated annealing* technique can be employed to deal with this problem, but it is very time consuming. It has also been found that the performance of the network is crucially dependent on the choice of the parameters (A, B, C, D). Only a good balance among the parameter values produces valid tours satisfying the constraints. Various modifications to the Hopfield-Tank architecture for solving TSP have been explored (see [9]).

8 Applications

Various ANN models and learning algorithms have been successfully applied to a large variety of problems belonging to the seven tasks shown in Figure 1. As pointed out in Section 1, one of the important applications of ANN is pattern classification. A pattern classification problem of high commercial importance is Optical Character Recognition (OCR). OCR deals with the problem of processing a scanned image of text and transcribing it into a machine readable form (for example, ASCII). The text may be machine-printed or handwritten. Actually, the term OCR is a misnomer as there is no "optical" processing involved in the transcription process. OCR is important in eliminating or minimizing the human labor involved in capturing information from paper documents. Two of the major application areas for OCR are in forms readers and text conversion in Digital Libraries. In this section we will outline the basic components of OCR and explain how ANNs are used for pattern classification.

The basic processing steps in an OCR system are shown in Figure 18. A paper document is scanned to produce a gray level or binary (black-and-white) image; a scanning resolution of 300 pixels/inch is typically used. In the preprocessing stage, filtering is applied to remove noise, and text areas are located and converted to binary (black/white) image using a globally or locally adaptive method.

In the segmentation step, the text image is separated into individual character patterns. This is a particularly difficult task for handwritten text where there is a proliferation of touching characters. It is also difficult for machine printed text when techniques such as "kerning" are employed. Noise could cause otherwise separated characters to be touching. Various techniques can be used to split composite patterns [5]. 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 pattern classifier.

Figure 19 shows the size-normalized character bitmaps of a sample set from the NIST character database [23]. We can see substantial intra-class variations. The goal of feature extraction is to extract the most relevant measurements from the sensed data, so as to minimize the within-class variability while increasing the between-class variability. Various feature

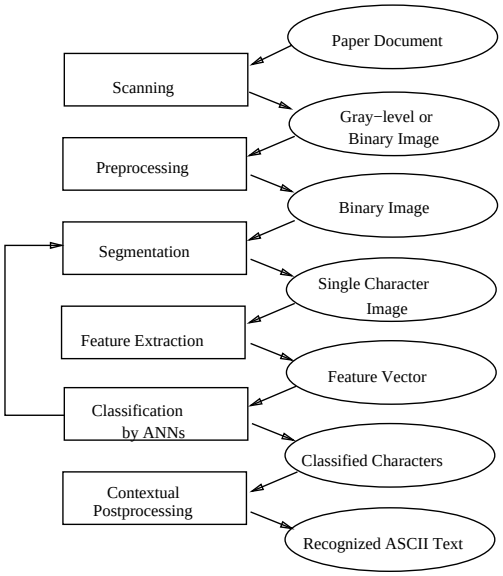


Figure 18: Diagram of a typical OCR system.

extraction methods have been employed for character recognition, including projection histograms, contour profiles, zoning, geometric moment invariants, spline curve approximation, and Fourier descriptors. There is no clear evidence as to which feature set is best for a given application. Figure 20 shows a typical scheme for extracting zone features [18]: contour direction and bending points. Contour direction features are generated by dividing the binary image array into rectangular and diagonal zones and computing histograms of chain codes in these zones, which results in 88 features (Figure 20(a)). Bending point features represent high curvature points, terminal points and fork points. A special geometrical mapping from bending points and their attributes to a fixed-length (96) feature vector has been designed (Figure 20(b)). The bending points in a normalized image are coded by positions which are quantized into 12 (4×3) regions, and by their curvature orientations which are quantized to eight (4 directions and convex or concave). The value of acuteness of a bending point is used as the magnitude for the corresponding component in the feature vector.

In the pattern classification stage, the extracted features are passed to the input stage of an ANN. The number of input units is equal to the dimensionality of the feature vector. The number of output units is equal to the number of character categories; for example,

Figure 19: A sample set of characters in the NIST data.

in the classification of numeral digits, 10 output units are required, where as for a mixed-case alphanumeric classifier as many as 80 units may be necessary (10 for digits, 26 for upper case, 26 for lower case, and about 18 for special symbols and punctuation marks). The number of units in the intermediate layer is usually determined experimentally so as to get the maximum recognition accuracy on an independent test set. In the OCR system described in [18], a two-layer feedforward network with 50 hidden units is found to produce good generalization ability.

Not all OCR systems explicity extract features from the raw data. A typical example is the network developed by Le Cun et al. [13] for zip-code recognition. The network architecture is shown in Figure 21. A 16×16 normalized gray level image is presented to a feedforward network with three hidden layers. The feature extraction implicitly takes place within the intermediate stages of the ANN. The 768 units in the first hidden layer form 12 8×8 feature maps. Each unit in a feature map is locally connected to a 5×5 neighborhood in the input image. All the units in a feature map share the same weight vector. Constructed in a similar way as the first hidden layer, the second hidden layer forms 12 4×4 feature maps. Each unit in the second hidden layer also combines local information coming from eight out of 12 feature maps in the first hidden layer. The third hidden layer consists of 30 hidden units. The 10 output units correspond to the 10 classes (digits '0' to '9'). The sub-network

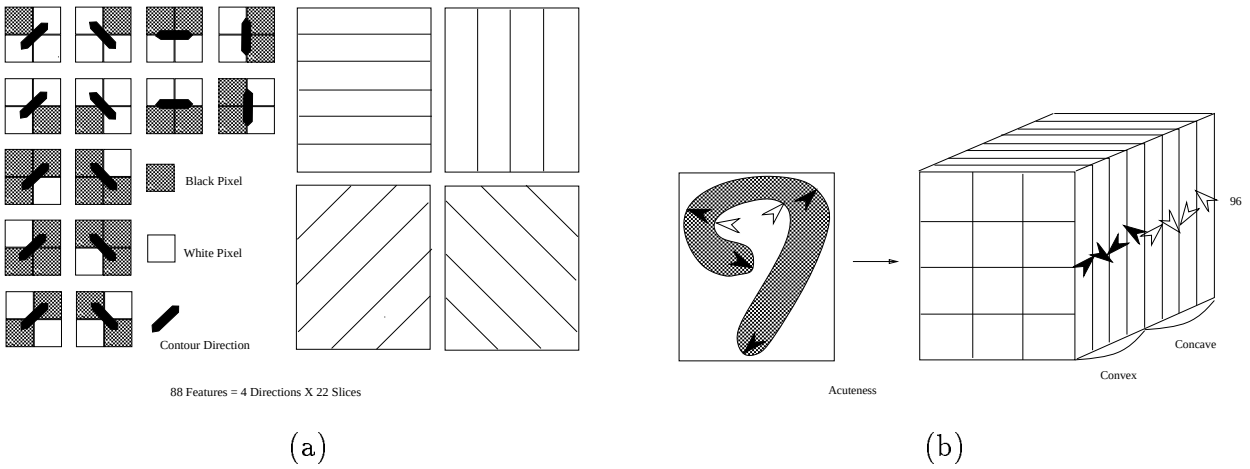


Figure 20: Zone features: (a) contour direction, (b) bending points.

from the second layer to the output layer is a standard fully-connected feedforward network.

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 exploited to update the output of the classifier. Examples are looking up in a dictionary of admissible words, or applying syntactic constraints such as for 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 in 1992 [23], more than 40 different handwritten character recognition systems were tested on the same 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 speed and in smaller memory requirements compared to nearest neighbor methods. Unlike the nearest neighbor methods, classification speed using ANN is also 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 is comparable to the human performance on isolated characters.

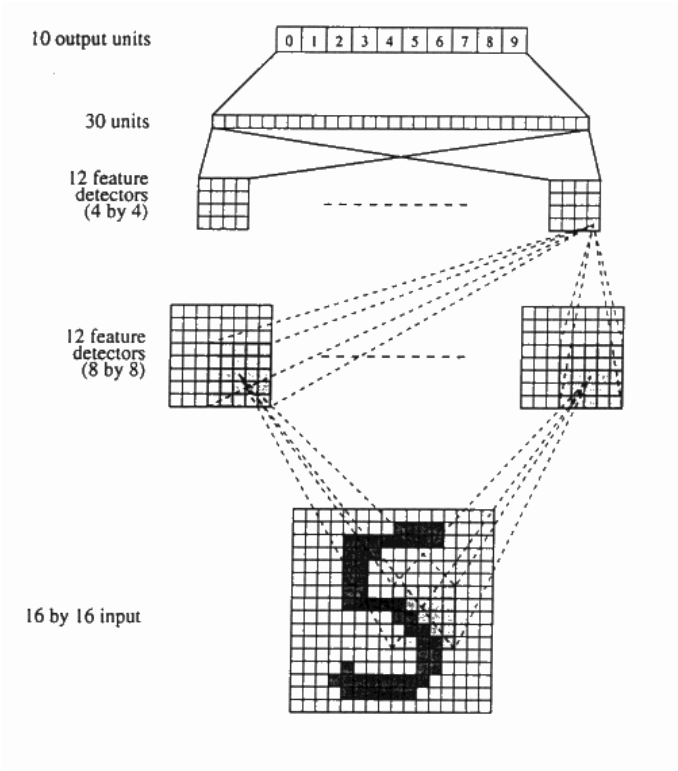


Figure 21: A network for recognizing numeric digits.

However, humans still outperform OCR systems on unconstrained and cursive handwritten documents.

9 Concluding Remarks

Developments in ANNs have experienced 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 developed in various disciplines, 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 disciplines in order to achieve a significantly better performance for challenging problems. Minsky [16]

has recognized that the time has come to build systems out of diverse components. In such a synergistic approach, not only are individual modules important, but also a good methodology for integrating various modules is the key to success. It is clear that communication and cooperative work between ANNs and other disciplines will not only avoid repetitious work but, more importantly, will stimulate and motivate individual disciplines.

Acknowledgment: The authors would like to thank Richard Casey, Chitra Dorai and Kalle Karu 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, Compters 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] R. G. Casey. Character segmentation in document OCR: Progress and hope. In *4th Annual Symposium on Document Analysis and Information Retrieval*, pages 13–39, Las Vegas, 1995.

[6] J. Feldman, M. A. Fanty, and N. H. Goddard. Computing with structured neural networks. *IEEE Computer*, pages 91–103, March 1988.

[7] S. Haykin. *Neural Networks: A Comprehensive Foundation.* MacMillan College Publishing Company, New York, 1994.

[8] D. O. Hebb. *The Organization of Behavior.* Wiley, New York, 1949.

[9] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley, Redwood City, 1991.

[10] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. In *Proc. Natl. Acad. Sci. USA 79*, pages 2554–2558, 1982.

[11] 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.

[12] T. Kohonen. *Self Organization and Associative Memory.* Third edition, Springer-Verlag, 1989.

[13] 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.

[14] R. P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, 4 (2):4–22, Apr. 1987.

[15] W. S. McCulloch and W. Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[16] M. Minsky. Logical versus analogical or symbolic versus connectionist or neat versus scruffy. *AI Magazine*, 65(2):34–51, 1991.

[17] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry.* MIT Press, Cambridge, MA, 1969.

[18] 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.

[19] N. J. Nilsson. *Learning Machines: Foundations of Trainable Pattern-Classifying Systems.* McGraw-Hill, New York, 1965.

[20] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

[21] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing: Exploration in the microstructure of cognition.* MIT Press, Cambridge, MA, 1986.

[22] B. Widrow. ADALINE and MADALINE – 1963. In *IEEE 1st Intl. Conf. on Neural Networks*, pages I143–I157, San Diego, CA, June 1987.

[23] 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, Gaitherburg, MD 20899, 1992.