CHAPTER 2

Artificial Neural Networks and Their Use in Chemistry

Keith L. Peterson

Department of Chemistry and Physics, Wesleyan College, Macon, Georgia 31210

INTRODUCTION

Chemists are being confronted with a confounding array of data at an ever-increasing pace. The advent of new experimental techniques, the development of cheaper, faster, and more precise instrumentation, and the availability of desktop computing power that only a decade ago would have filled a small house have all contributed to this situation. Medicinal chemists using combinatorial chemistry methods have generated huge libraries of chemical compounds that must be assessed for pharmaceutical activity. Spectroscopists search and analyze huge databases of spectra. Computational chemists generate vast numbers of points describing potential energy surfaces in n-dimensional spaces.

The issue facing chemists today is not how to generate data, which not so long ago was actually quite difficult and time-consuming, but how to extract useful information from the data generated. As a consequence, a branch of chemistry known as chemometrics began to evolve in the late 1960s. Chemometrics is broadly concerned with the extraction of useful information from chemical data. Until about 1990, chemometrics primarily involved appli-
Artificial Neural Networks and Their Use in Chemistry

cation of classical statistical methods such as pattern recognition, discriminant analysis, regression analysis, and principal components analysis to chemical problems. (It is not our intent to discuss these methods in any detail; the reader is referred to any number of excellent texts.)

In the late 1980s chemometricians and other chemists rediscovered another tool, the artificial neural network (ANN) which, as it has turned out, has made a significant impact on the war on data interpretation. We say "rediscovered" because the idea for ANNs germinated in 1943. Before continuing with a brief introduction to ANNs, it is worth noting that although chemometrics was once the province of analytical chemists, this is certainly not the case today; virtually every branch of chemistry has been affected by this discipline.

Overview and Goals

This chapter is intended to introduce you to ANNs and their applications to solving chemical problems. The only prerequisites are some chemical knowledge and a bit of calculus (you should know what a derivative is). Our goals are the following:

1. To start from "ground zero" and bring you to the point of being able to intelligently assess the literature on chemical applications of ANNs.
2. To help you develop a good sense for how ANNs may be applicable to particular problems.
3. To introduce a number of the different types of ANNs (e.g., backpropagation) by telling you how each one functions, for what types of data analysis it is useful (and also not useful), and to what chemical problems it has successfully been applied.
4. To provide a user's guide for the application of neural network software, be it commercial or in-house. This guide should help you decide which type of network to use, what to do with your data before submitting it to a neural network, how to assess your results, and what to do when things go wrong; i.e., when it appears that a neural network cannot solve your problem.

Although we are interested in how ANNs relate to chemical problems, we will discuss several nonchemical examples. The reasons for this are twofold: certain concepts are more easily understood with these examples, and they serve to illustrate that there is a large body of knowledge concerning ANNs that has not yet been applied to chemical problems.

The field of ANNs is vast and no one chapter on the topic can be totally comprehensive. A multivolume text would be required just to cover the chemical applications of ANNs. So, what will we leave out? First, we will be short on theory, though not necessarily equations. Often we will be quite qualitative in
describing why a network functions as it does, and derivations will be kept to a minimum (i.e., zero). In our experience, to successfully apply an ANN to a particular problem, you need to know what it is doing and how/why it is doing it, but you usually do not need to be able to derive the equations governing the network. There are many excellent texts that extensively treat ANN theory. A few of our favorites are given in Refs. 10–16. This is not an exhaustive list, and a trip to a library (or a database search!) will uncover others.

Second, our review of the literature will not include any applications to chemical engineering or material science; these have been covered elsewhere.17,18 Most chemical applications cited here are from 1992 to early 1999. The reader is referred to several excellent reviews that cover previous time periods.19–32 We have tried to be comprehensive within the chosen time period in reporting the different types of chemical application and the different types of ANN used in these applications. However, if 45 references were found in which a backpropagation network was used to optimize liquid chromatography parameters, we include here only the later ones, plus those that we felt were unique or different, or contained otherwise interesting conclusions. Nonetheless, from these cited papers it should be possible to gather an extensive list of references in a particular specialty, if necessary.

Third, we will not discuss every type of ANN, nor list every learning rule, transfer function, etc. that has ever been used. It is impossible to do this because variations on a theme (and there are many themes here) are constantly being developed. We have, however, compiled rather extensive lists of learning rules and useful functions. We have tried to discuss all of the different types of network for which we have found referenced chemical applications. We have also included some networks for which we found no applications but believe offer strong potential for solving chemical problems. Some network types, particularly backpropagation, have many variations; for some of these variations we have limited our discussion and simply given a reference.

The next section introduces ANNs, stating in general terms what they are and how they operate, and establishes some common terminology. It ends with a brief history of ANNs. The second section outlines the types of problem ANNs can be applied to, and their relative strengths and weaknesses in comparison to statistical and artificial intelligence methods. In the third section, we give a detailed description of processing element operation and include extensive lists of learning rules, as well as summation, transfer, error, and output functions. Then in the next two sections, we look at collections of processing elements and tie everything together with descriptions of various network types and a review of chemical applications. Sections near the end of the chapter address practical considerations such as how to set up the network, how to preprocess data, and how to assess network performance. Finally we briefly review ways in which networks can be analyzed to discover how a particular network makes its decisions.
WHAT ARE ARTIFICIAL NEURAL NETWORKS?

Analogy with the Brain

For pedagogical purposes, it is useful to make an analogy between the brain and an ANN. The following simplified overview of brain structure at the cellular level suffices to make the key points.

The neuron, the basic cellular unit of the brain, consists of three parts: the soma, or body; the dendrites, which are relatively long spindly structures leading into the soma; and the axon, a single spindly structure that leads out of the soma and eventually branches. Each end point of the axon belonging to one neuron eventually comes into contact with an end of a dendrite belonging to another neuron. There are several end points of the original axon, and each one may contact a dendrite belonging to a different neuron. In this way, the original neuron may be connected to other neurons. Each of these other neurons may be connected to different sets of still other neurons, thereby forming a network of neurons, i.e., a neural network. Nerve signals (information) may propagate from one neuron to another via the axon–dendrite connections.

But, we have been too cavalier with the words “contact” and “connection.” The contact between an axon and a dendrite is not a direct contact; there is a small gap between them, which is called the synaptic junction or synapse. An electrical nerve signal moves down an axon to the synapse, where it is converted to a chemical signal, which crosses the gap via a neurotransmitter such as acetylcholine. The chemical signal is converted back to an electrical signal when the neurotransmitter reaches the dendrite side of the gap and is absorbed. The signal can go in only one direction—axon to dendrite—because the dendrite cannot release neurotransmitters.

A moment’s thought should convince you that there must be more to it than that. First, there must be a way of getting the signal into the brain and also a way of getting it out; otherwise a signal would literally bounce around continuously. Input and output mechanisms are typically sensory in nature (sight, smell, touch, etc.). It would lead us too far astray to discuss this, and we simply note that input and output mechanisms are necessary. Assuming that these are present, there still must be other processes occurring. A given neuron may be connected to many other neurons, giving rise to the possibility that the input signal to the neuron is very large. It seems reasonable then, that either the synapses or the neurons themselves, probably in the soma, modulate these signals. Biological evidence indicates that signals are modulated in both places. We will defer the effect of the soma until later, noting here simply that the amount by which a signal is changed at or by the synapse depends on the synaptic strength. Synapses have the ability to adapt their strengths in response to various stimuli (i.e., signal inputs) and this is the essence of learning.
What Are Artificial Neural Networks?

Artificial Neural Networks

ANNs were originally motivated by attempts to mathematically describe or model how individual neurons and networks of neurons operate. It can be illuminating to keep the analogy between the brain and an ANN firmly in mind, since much current research in ANN theory is still biologically motivated and the analogy can be very helpful in understanding this work.

We now flesh out the analogy in general terms. We will become much more detail-oriented in the sections on Processing Elements, Collections of Processing Elements, and Different Types of Artificial Neural Network. Before continuing, we will briefly comment on the word "artificial." This word merely indicates that we are, and will be, talking about neural networks that are mathematical models of biological neural networks; these models will be implemented on computers. Some workers prefer the word “computational,” although most settle simply for the phrase “neural network.”

In an ANN, a processing element (PE) plays the role of a biological neuron. A PE is synonymous with “node,” “processing unit,” “unit,” and “neuron,” the latter taken in the artificial context. We will use “PE” throughout this chapter. Each PE has an output connector that plays the role of an axon. As such it will branch and connect to other PEs. These connections are really not to PEs per se, but rather to input connectors that play the role of dendrites. The connections play the role of synapses. A single PE with two input connectors and one output connector is shown in Figure 1a. In our diagrams information flows from bottom to top unless otherwise indicated. We will indicate a branched output connector as shown in Figure 1b. The branch occurs at the juncture of the PE and the output connector, forming a V at the top of the PE. This PE can be connected to four other PEs, as shown in Figure 1c.

To continue our analogy we must amplify our earlier statement that the “connections . . . play the role of synapses.” This is certainly true, but if we

![Figure 1](image-url)
remember that synapses modulate signals and that their ability to adapt their strengths constitutes learning, then we must find a way to account for this. The analog to the synaptic strength is the weight of the connection, or simply weight. Every connection in our ANN has a weight associated with it. (The weights are not explicitly shown in our diagrams.) This weight is simply a number, but a very important number, because if we can find a way to make it change in response to information passing through the connection with which it is associated, we will have done what a synapse does when it adjusts its strength in response to signals that pass through it. In other words, we will have caused our ANN to learn! There are various ways to accomplish this (we use so-called learning rules) and these are discussed in subsequent sections.

We have said very little thus far about what a PE does other than that it must modulate information passing through it. For the moment, we will simply think of the PE as a black box. Information passes into it via connections with other PEs and then passes out of it via connections after having been modulated or acted upon mathematically. The details of these mathematical operations are deferred to a later section.

Finally, we need input and output mechanisms for our ANN. To get information into an ANN we will use input PEs. An input PE has only one input connector but may have an arbitrary number of output connectors. The output connectors will connect to other PEs, but the input connector will not connect to any other PEs. An example of an input PE having one input connector and two output connectors is shown in Figure 1d. To get information out of an ANN we will use output PEs that have only one output connector (not connected to any other PEs) but an arbitrary number of input connectors. An example is shown in Figure 1e; only two PEs may be connected to this output PE, and these are via the two bottom input connectors. We note that the input connector on an input PE and the output connector on an output PE are not parts of bona fide connections between PEs. As such we might think that there will be no weights associated with them. For mathematical purposes, however, the connectors are usually assigned weights of 1.0 and are kept from changing; that is, they have fixed weights and do not participate in learning. Also, input PEs typically do not modulate signals, although output PEs typically do. Putting all our ideas together, we can construct a diagram such as the one in Figure 2. Such a diagram is a representation of an ANN.

Summary of Neural Network Operation

The ANN in Figure 2 is admittedly quite simple. Nonetheless it will allow us to illustrate many of the qualitative features of ANNs and their operation.

We begin by making some observations about the architecture, that is, the arrangement of the PEs and the manner in which they are connected in the ANN. This is the structure or the topology of the network. First, the PEs are arranged in a series of layers. The network in Figure 2 consists of three layers of
What Are Artificial Neural Networks?

Figure 2 A simple three-layer artificial neural network (ANN).

PEs: the input layer, the middle layer, or synonymously, the hidden layer (HL), and the output layer. There may be more than one HL in a network. This network could be referred to as a three-layer network. But, not everyone uses this terminology. The same network could be called a one-layer network, a two-layer network, or a three-layer network! Those who use a “one-layer network” are simply casting the input and output layers aside, regarding them as uninteresting. In a sense, these people are correct, since most of the interesting aspects of network operation take place in the hidden layer, or layers. Those who refer to the network of Figure 2 as having two layers usually take these layers to be the hidden and output layers. The apparent rationale here is that the interesting stuff takes place in the hidden layer, and it is important to get the results of this interesting stuff out of the network via the output layer, the input being rather trivial. We shall refer to the network of Figure 2 as a three-layer network, not only for the obvious reason, but also because many ANN computer software packages use this terminology. When reading the ANN literature, it is essential to be aware of how the author defines a “layer.”

Second, the PEs are connected to each other in a particular fashion. In our example each of the input PEs is connected to both HL PEs. In contrast to this, each of the HL PEs is connected to only one output layer PE. There are different types of ANN, and each has its own connection scheme that specifies which PEs are connected to each other.

Third, each layer has a certain number of PEs. In our example each of the three layers happens to have two PEs. The number of PEs in each layer depends on the type of network and the data being used.

We next need to describe the flow of information through the network. This will also serve to introduce some concepts that are central to the application of ANNs to the analysis of chemical data. Suppose we have two numbers, 0.15 and 0.43, that we want to use as input to our ANN in Figure 2. Let the 0.15 enter the network through the input connector to the PE labeled A and the 0.43 enter the network through the input connector to B. What happens to these two numbers? The 0.15 passes through A unchanged. (Recall that an
input PE and its connector usually do not modulate signals.) But then it " splits": 0.15 goes to both C and to D. From C the input goes to E and then out of the network; from D the input goes to F and then out of the network. Of course, the input is not 0.15 when it comes out of the network at either E or F, since it has been modulated by the weights and PEs through which it passes. The 0.43 passes through the network in a fashion completely analogous to the passing of the 0.15, the only difference being that its path is slightly different because it entered the network at a different location. The 0.15 and the 0.43 are passed through the network simultaneously or in parallel, and therefore really lose their identity, particularly in the middle layer where, for example, C is receiving both the 0.15 and the 0.43 and then mixing these two signals together. We say that the information which went into the network is distributed throughout the network. Even though we have referred to the inputs as 0.15 and 0.43 everywhere in their trip through the network, these values do not really exist as such. We referred to numerical quantities not to be confusing, but simply to stress the flow of information. Also, we do not really care at the moment what the weights are or how the modulation occurs; we will get to that later.

Now suppose that the output of the network is the two numbers 0.35 and 0.85; the network has mapped the pair of numbers (0.15, 0.43) to the pair of numbers (0.35, 0.85). In very general terms, an ANN may be thought of as a mapping device. Of course in some sense you do not really care that (0.15, 0.43) gets mapped to (0.35, 0.85): suppose that your data contains (0.15, 0.43) but that this pair is not associated with (0.35, 0.85) but rather with, say, (0.30, 0.86). What can you do? Well, if we can determine a way to change the weights in the ANN, we can make the ANN learn. Let us assume that we know how to change the weights. This is tantamount to knowing the so-called learning rule and a few other things to be discussed shortly.

Now let us start the whole process over. We enter (0.15, 0.43) into the network as before but also present the desired output (0.30, 0.86) to the output PEs (0.30 to E and 0.86 to F). Suppose the actual output is (0.35, 0.85), as before. But now the network knows how to change the weights (we have told it how to do this), and it also knows that the output should be (0.30, 0.86). We then pass (0.15, 0.43) through the network again, after the weights have changed, and find that the actual output is (0.33, 0.83). This is still wrong (brains do not always learn the first time, either), so the weights are changed again and the inputs passed through the network once more, this time giving an output of (0.31, 0.85). Still wrong, so repeat the whole cycle again, this time with an output of (0.30, 0.86). The network has learned the proper output! The network has learned to multiply 0.15 and 0.43 by 2.

Real-life data that is interesting is not as trivial as the example above. But much data does exist that can be split into a set of independent variables (the inputs to an ANN) and a set of dependent variables (the desired outputs of an ANN). Suppose you have a data set with five independent variables and one dependent variable, and you know through some combination of experiment and computation the values of all six variables for, say, 50 cases. [In our exam-
What Are Artificial Neural Networks?

ple, we had two independent variables whose values were 0.15 and 0.43, and two dependent variables whose values were 0.30 and 0.86. We had a case that could be represented by the ordered quadrant (0.15, 0.43, 0.30, 0.86). If we have 50 cases, then we have 50 ordered quadruples.

Now suppose further that there is a 51st case for which you know the values of only the five independent variables, and we would like to predict the value of the corresponding dependent variable. You could generate the value, but this might not be feasible for a variety of reasons, time and cost among them, especially if you anticipate having to do this many times. You might reasonably assume that if you could train an ANN to learn your data (the first 50 cases), the network might be able to do a reasonable job of predicting the dependent variable for the 51st case. That is, you assume that since brains are capable of predicting or generalizing from learned information, perhaps the trained ANN will also have some generalizing capability. (This can, under certain conditions, be a dangerous or invalid assumption, but for now we will accept it and postpone a discussion of exceptional conditions.)

So you train a neural network consisting of five input PEs, probably five to ten HL PEs and one output layer PE. You pass the independent variables for the first case through the network while presenting the dependent variable for the first case to the output PE. The weights, which were probably initially set to random values between, say, +0.2 and -0.2, may change (depending on the network type) after this. Do the same thing for the second case (the weights may change yet again), third case, . . . , 50th case. At this point you have passed the entire data set through the network once; that is, each of the 50 cases has passed through the network one time. This is one training epoch, or just epoch. This is probably not enough for the network to learn the data: 50, 100, maybe even 1000 or more epochs may be required.

Assume that the ANN is trained after 100 epochs. Introducing some terminology, we say that the ANN underwent supervised learning or supervised training. Supervised learning requires that the desired output or desired response be provided to the network by a “knowledgeable teacher,” in this case, you. Each of the 50 cases is called a training vector or training pattern. The entire set of 50 training patterns is called the training set.

Now, you want to use your trained network to predict the dependent variable value for the 51st case—a pattern that is not in the training set and consists of a value for each of the five independent variables. Simply apply these five inputs to the input PEs and observe the output at the output PE. This is the predicted value. In this phase of operation the network is in a test mode or recall mode, as opposed to a training mode.

At this point we must address two questions: What happens if the network cannot learn the training data? And, how good (accurate, reliable) are the predicted results? In response to the first question, we note that usually a network cannot or will not completely learn the training data! Very often, this turns out to be inconsequential because as a rule there is experimental error or uncertainty associated with data, both input and output. Also, it often happens
that networks that have not learned training data exactly are better able to predict or generalize than those which have learned the training data. It is better that the ANN not be overtrained. This point leads to the second question. Instead of using all 50 training vectors, you may want to use only 40 to train the network, treating the remaining 10 as test patterns, i.e. a test set. The trained network is presented with the test set in test mode. Comparison of actual outputs for the test set with the desired outputs will give you a good idea of how good the network's predictions on actual test patterns are likely to be. We will have more to say about this later.

We have made some assumptions about how our example network functions. Many types of ANN operate as we have assumed, but some do not, and we now indicate these differences. The just described ANNs are heteroassociative because the desired outputs differ from the inputs. When the desired outputs are the same as the inputs for all the training vectors, the network is autoassociative. This circumstance naturally requires that the number of input PEs be equal to the number of output PEs. Some types of network—for example, backpropagation—may be configured as either hetero- or autoassociative, whereas other types must be heteroassociative, and still others must be autoassociative.

If no desired output is presented to a network, then learning that occurs is unsupervised. Some types of network are capable of supervised learning only, and other types are capable of unsupervised learning only. If there is a knowledgeable expert that indicates only whether a network output is good or bad, learning is said to be reinforcement learning. This is somewhere between supervised and unsupervised learning.

An ANN must have at least one learning rule. Learning rules specify how weights change in response to learning inputs. Learning may require many epochs of the training data or only one. Finally, learning rules may contain parameters that need to be changed as training occurs.

Our network above is a feedforward network because information flows only in the direction from the input layer to the output layer. We could have turned them into feedback networks by connecting the output PEs back to the hidden layer PEs. In this situation, information may get passed from the hidden layer to the output layer, back to the hidden layer, back again to the output layer and finally, out of the network. The feedback connections, of course, have weights associated with them, and these weights are changed by a learning rule. Typically, information bounces between two layers until some preset convergence criterion (such as average output error being less than some tolerance) is met. In addition to backward connections, it is possible for PEs within a single layer to be interconnected. These lateral connections also have weights that are affected by learning rules. PEs may also be connected to themselves.

We assumed in our network that what happened to one input happened to the other input as well, and, in particular, happened at the same time. For example, we assumed that both input values entered the network at the same time, both left their respective input nodes at the same time, were modulated at
the same time, and so on, until finally, two outputs left the network, also at the same time. Such a network operates in a *synchronous* mode. In an *asynchronous* mode PEs usually release their output values at random and also independently of any other PE in the network. (Note: In an ANN implemented on a serial computer, only one operation at a time is performed. For a synchronous ANN, the first input variable will be read into the first PE, the second variable into the second PE, etc., in a serial fashion. Even though each read occurs at a physically distinct time, we consider them to have all occurred at the same time.)

**Brief History of Neural Networks**

Having given a basic picture of how an ANN operates, we now give a short history of the development of ANNs. This will serve to explain why it has taken 40–50 years for this field to blossom. Along the way we will introduce some of the important people who were responsible for the beginnings, initial development, subsequent decline, and finally, revitalization of ANNs. We will certainly not mention all the important workers; no slight is intended to any of them. The reader who is interested in the historical aspects of ANNs is referred to the appendix of Simpson’s book.14

In 1943 Warren McCulloch and Walter Pitts (a neurobiologist and statistician, respectively) made the first mathematical model of an ANN.9 This created great excitement because it showed that an ANN could generate sophisticated results from a series of simple computations. Unfortunately, it lacked a key ingredient of ANNs as we now know them, the ability to learn. In 1949 Donald Hebb described a concept that eventually led to a mathematical procedure for learning, that is, a learning rule.33 Several learning rules now bear his name. In the late 1950s Frank Rosenblatt at Cornell generalized the work of McCulloch and Pitts by incorporating learning into their model.34,35 Rosenblatt’s creation was called the perceptron. He was able to prove that a two layer perceptron could solve exactly certain types of linear problem, but he could not find a mathematical way to adjust weights of HL PEs in his three-layer perceptrons. In 1959 Bernard Widrow of Stanford introduced a variation on the perceptron called the adaline,36,37 utilizing what is now known as the Widrow learning rule. Like Rosenblatt, however, he could prove only that the model could solve certain types of linear problem. In the mid-1960s Marvin Minsky and Seymour Papert at MIT’s Research Laboratory of Electronics began an in-depth study of perceptrons.38 They demonstrated many limitations of two-layer perceptrons, in particular the inability to solve nonlinear problems, and were unable to demonstrate that three-layer perceptrons could overcome any of them. The MIT workers published their finding in a book38 that contained a statement to the effect that perceptrons in particular and ANNs in general were no longer worth studying. This conclusion had a drastic effect on ANN research, the result being that funding and the number of researchers dropped dramatically.
Of the researchers who remained active, among the most notable are James Anderson, Tuevo Kohonen, Stephen Grossberg, Terrence Sejnowski, Robert Hecht-Nielsen, and the group headed by David Rumelhart and James McClelland. Anderson at Brown was responsible for the brain-state-in-a-box (BSB) ANN. Kohonen of Helsinki Technical University was concerned with PEs that competed with each other for the privilege of learning. This resulted in the learning vector quantization and self-organizing map ANN. At Boston University’s Center for Adaptive Systems, Grossberg applied results of physiological studies to arrive at many important ANN results, among them adaptive resonance theory networks. Sejnowski at the Salk Institute invented the Boltzmann machine ANN and has made significant contributions to the application of backpropagation networks including their use in predicting the structure of globular proteins. Hecht-Nielsen founded neurocomputing research and development programs at Motorola and TRW, developed the counterpropagation ANN, and proved that there exists a three-layer ANN that will perform any continuous mapping defined on a convex set. McClelland of Carnegie-Mellon and Rumelhart of Stanford produced a variety of important ANN results that resulted in a standard book.

In 1982 John Hopfield, a physicist at California Institute of Technology, presented a paper to the National Academy of Sciences that was the first paper concerning ANNs this group had heard since the 1960s. Most observers credit Hopfield’s enthusiasm and clarity of presentation with prompting a significant number of researchers to become interested in ANNs once again.

**WHAT CAN NEURAL NETWORKS BE USED FOR AND WHEN SHOULD YOU USE THEM?**

Neural networks have been used to solve a wide variety of data analysis problems. In order to put these applications in perspective, it is useful to categorize them in some fashion. Generally, we recognize four categories which are almost mutually exclusive: classification or categorization, modeling, associations, and mappings. The associations we have in mind here are mainly autoassociative, and as explained shortly, we use the word “mappings” in a sense different from that in the Introduction. Although these categories are virtually exclusive of one another, you may find that to solve a particular problem you need to use methods from more than just one of them.

**Classification**

Problems of classification arise frequently. For example, is the secondary structure of a particular protein an α-helix, a β-sheet, or a coil structure? What
is the geographic origin of a pure olive oil sample? Given the infrared spectrum of a compound, what structural features does the compound contain?

The first question obviously poses a classification problem. The second question is more subtle: What exactly do we mean by “geographic location”? If we mean latitude and longitude, then we would have difficulty with classification because these are continuous variables; categories are discrete. If, however, we mean a region (which does not overlap with any other region) of Italy such as Sicily or coastal Sardinia, then we have a well-defined classification problem. The third question is perhaps subtle, depending on your viewpoint, but also different in an important respect. A protein segment cannot exist as both an α-helix and a β-sheet, and a pure olive oil cannot originate from two locations. But most molecules have more than one structural feature; if we think of these features as categories, then a molecule can belong to more than one category.

How do we go about solving these problems? We can't answer this until we know more about the data we have. Consider the olive oil example. 572 olive oils from nine different regions of Italy were analyzed for the percentage of eight different fatty acids. If we use ANNs, we will use eight input PEs (one for each fatty acid) and probably nine output PEs (one for each region). The nine different regions constitute nine categories. If we let each region be denoted by either a 1 or a 0, a 1 indicating the oil is from that region and a 0 indicating the oil is not from that region, then an oil from region three would have desired outputs of 0, 0, 1, 0, 0, 0, 0, 0, 0 and 0 at the nine respective output PEs. We would train our network to learn 572 mappings of eight inputs to nine outputs.

A typical mapping that we want the network to learn might be (5, 10, 25, 40, 5, 5, 5, 10) → (0, 0, 0, 0, 0, 0, 1, 0, 0). Of course, we might want to use say, 489 cases to train the network and the remaining 83 cases as a test set. For the moment, we have made the points we wish to make. The interested reader is referred to Ref. 19 for a complete analysis.

There are a few things we should mention about this example and to which we will return later in the section on Practical Considerations. First, we would be courting disaster if we actually entered numbers like 5, 10, 25, or 40 into our network. Input data typically needs to be scaled so that it fits in a narrow range, usually 0 to 1. Second, why not use one output PE and let the desired outputs be integers from 0 to 8, or 1 to 9? Scaling is again part of the problem, but there are other problems as well. Third, we noted in the Introduction that 100% learning is rarely accomplished. What do we do if the best actual output is (0, 0, 0, 0, 0, 0, 0, 0, 0.4) for a desired output of (0, 0, 0, 0, 0, 0, 0, 0, 0)? Or if the best actual output for the same desired output is quite different, such as (1, 0, 0, 0, 0, 0, 0, 0, 0). Unfortunately no matter what we do we will not achieve 100% training. Finally, what type of network (backpropagation, counterpropagation, etc.) should we use, and how many hidden layer PEs should it have?

Next, let us briefly consider the infrared spectrum example. Again, for full details the reader is referred to Ref 19. Here, we wish only to make a few points analogous to those just made for olive oils. The data consists of 2499 spectra, each of which is divided into 256 intervals between 4000 and 400 cm⁻¹. If a
peak is found in a given interval, its intensity is scaled to lie between 0.0 and 1.0. If there is no peak in an interval, the intensity for the interval is 0.0. The structure of the compounds is described in terms of the presence or absence of 36 functional groups. The desired output for the compound consists of a string of 36 digits, each a zero or one, where zero indicates the absence of a functional group and one indicates the presence of a functional group. The neural network would have 256 input PEs and 36 output PEs. Note that in contrast to the olive oil case, a desired output here can have more than one “1,” reflecting multiple category membership. The comments and multiple questions raised about the olive oil network in the preceding paragraph could be made here too.

We now offer some general observations about these examples.

1. The relationships between the input data and category membership are quite complex. There are no explicit equations that allow the calculation of geographic origin from the percentage composition of eight fatty acids. A similar statement holds for the infrared spectra example. (The cynic might say that it is possible to calculate an infrared spectrum of a molecule from quantum mechanics. We cannot deny this but would claim that to calculate with sufficient resolution the spectra of 2499 relatively large molecules is, for all practical purposes, prohibitive from both time and cost points of view.)

2. How did we know that knowledge of fatty acid composition would allow a network to learn where an olive oil came from? Or, that assigning an intensity to 256 intervals would allow a network to learn what structural features a molecule has? Put very simply, we did not. We were testing a hypothesis, according to the scientific method. Although it might seem plausible that olives grown in different regions might have oils with different compositions (owing perhaps to different soils, temperature, rainfall, etc.), it does not seem a priori evident that this should give rise to classification criteria of some (or any) sort. It could have turned out that as far as the neural network was concerned, fatty acid composition had nothing to do with classification, but perhaps trace heavy element composition did. So, what might you do if you cannot train a network satisfactorily? If you had sufficient resources you might attempt to get more data, either percentages of other fatty acids, or heavy element composition, or something else. Or, you might try to preprocess the inputs in some way other than simply scaling to values between zero and one. Or, you might try a different classification method, either another type of neural network or perhaps a method from statistics, such as discriminant analysis. This leads us to our next point.

3. How did we know that the type of network we chose (it was backpropagation) or for that matter, any neural network, would be able to learn a classification criterion? Again, simply put, we did not. Perhaps a different type of network would give better training, or perhaps none would. However, some types of network are designed to do certain things well, and after enough successful case studies have been compiled, an empirical body of knowledge develops that can serve effectively in guiding you to a reasonable network.
What Can Neural Networks Be Used for?

choice. When we discuss different network types later in the chapter, we will indicate the sorts of problem to which they have been applied with success. You should begin to see here that there can be, and often is, a significant interplay between input data, how it is preprocessed, and choice of network type.

The alert reader will have noticed that outputs have not been discussed relative to success, or lack thereof, in solving classification problems. In fact, how you choose categories can be critical. If you are beginning to think that data analysis with neural networks is a no-win situation, do not despair; such problems can be very difficult regardless of the method you choose, and you have, as it turns out, as good or better chance of success with neural networks as with any other method! Perhaps it is just not possible to distinguish Sicilian olive oils from coastal Sardinian ones, no matter how much data of any kind you may have; perhaps these two categories are not "natural" categories of your data. Well, such is life, and if you have exhausted all other possibilities and find that no matter what data you use, how you preprocess it, or what classification method (neural network or not) you use, you still cannot discriminate between oils from these two regions, then there really is not much you can do to change your predicament.

An apparently worse predicament is to have no idea of what the categories should be for a particular problem. Then what? You perform a cluster analysis; that is, you use a neural network to see if your data clusters or forms categories and if so, what those categories are. Let us return to the olive oil example and assume that we have the same data except that we will pretend we do not know what the categories are. Suppose we have a network that is capable of unsupervised learning. We will assume that this network has only two layers: an input layer of eight PEs and an output layer. As we will see, the output layer will form categories and may have an arbitrary number of PEs, depending partly on how many categories are desired. We would train this network as before except that there would be no desired output shown to it. If we monitor the weights to the output layer PEs of the network as training proceeds, we will find that eventually these weights essentially stop changing. At this point, the network is trained. This criterion for stopping training is different from the one we used before. This criterion can be used for some supervised learning networks, although it is more convenient just to compare actual and desired outputs for these networks. In the unsupervised case, we do not know what the desired output is, so we need a different criterion.

Now we take our trained network and pass the training set through it, noting which output layer PE has the highest output for each input. If there are fewer output layer PEs than there are training set cases, then we will find that more than one input will cause a given output PE to have the highest output value. We can consider that this given output PE represents a category whose members are the input cases that caused it to have the highest output. If the output layer has nine PEs, we would be forcing the network to form nine
categories. Unfortunately, the members of these nine categories are not the same as the members of the original nine categories based on geographic location. We need a network with more than nine output PEs. This obviously gives us more than nine categories, but for enough PEs we would find that, for example, three of them taken together might correspond to "Sicily." For a slightly different viewpoint on this, see Chapter 10 of Ref. 19.

The utility of a network like the one we have been discussing, which is similar to a Kohonen or self-organizing map network, is that it lets you determine the category structure of a data set. This is invaluable if you have no a priori idea of what this structure is. Simply train some networks with different numbers of output layer PEs. This will let you see which cases are assigned to particular categories and how cases shift into other categories as the number of output layer PEs change.

As with the supervised learning categorization networks, there are a few items that need to be discussed for the unsupervised case. Scaling or preprocessing of input data is still important. The number of output PEs is usually arbitrary unless you have reason to believe your data should fall into a certain number of categories. The issue of whether a network can be trained at the near-100% level is irrelevant because you do not know what the correct answers are. It is possible to use an unsupervised learning network to classify data whose correct classifications are known. In this case you can talk about percent correct; again, see Chapter 10 of Ref. 19.

Now, why would you use an ANN to classify objects when well-known statistical methods such as discriminant analysis can do the same thing? Most statistical methods rely on matrix algebra, and for large data sets the matrices involved can become quite large. Of course, large data sets will generally require networks with more PEs (and therefore weights). So, in both cases computation time will increase with the size of the data set. Often, however, the larger the data set the more efficacious ANNs become. The drawback here is that if you use a network type that has no prescription for the number of PEs to use in a given layer (typically a hidden layer), you may spend a lot of time training different networks to find one that gives an acceptable level of training and prediction performance. We will return to the issue of determining the best number of PEs later in the section on Practical Considerations. For now, we can say that the smallest network consistent with your training and prediction requirements is the best network to use.

Most, though not all, statistical methods are linear. (Basically, in linear methods, data variables are raised only to their first power and are neither multiplied nor divided by other variables. A simple example is the linear regression line \( w = ax + by + cz \), where all the variables are raised to the first power. The equation \( w = axy + bz^2 \) is nonlinear in \( x, y, \) and \( z \) because the variables \( x \) and \( y \) are multiplied by another variable—\( y \) and \( x \), respectively—and because \( z \) is raised to a power other than one. This equation, however, is linear in the new variables \( xy \) and \( z^2 \).) In the case of discriminant analysis, this means that categories should be linearly separable. If they are not, it will be impossible for
What Can Neural Networks Be Used for?

Discriminant analysis to train to the 100% level. Most ANNs, however, are nonlinear (due to nonlinear functions; see the later section on Processing Elements), which means that they should be able to train at or very near the 100% level regardless of whether categories are linearly separable. Simple illustrations of a linearly separable and nonlinearly separable data set are shown in Figures 3a and 3b. To illustrate the concept, only two independent variables (2 inputs) are shown. It is clear that a straight line completely separates the members of category 1 from those of category 2. Presumably these members are part of a training set and assuming that they are representative of the entire population of members, categories 1 and 2 are truly linearly separable, and either discriminant analysis or a neural network should give satisfactory performance. Figure 3b illustrates two categories that are not linearly separable, and you might expect to be able to find an ANN that performs better than discriminant analysis.

Discriminant analysis works best when the independent variables are uncorrelated (i.e., orthogonal), whereas neural networks are relatively insensitive to this. However, using several variables which are correlated may cause your neural network to be larger than it really has to be, which can be a drawback for large data sets. You are advised to check your variables for correlations and consider removing them with a method such as principal components analysis (PCA). Without going into details, this method basically replaces your original variables with a new set, which is uncorrelated. (PCA is different from principal components regression. In PCR, the new variables formed from PCA are used in a regression equation.) If you need to be able to explain what attributes an object has that cause it to belong to a given category, then you should strongly consider a statistical method such as discriminant analysis, which can tell you how important each independent variable is in determining the object's category membership. (It is possible, although more difficult, to use an ANN to determine a variable's importance. We discuss this in the last section of this chapter.)

The caveat here is that discriminant analysis, like most statistical methods, relies on certain assumptions about your data (random errors, normal distribu-
Artificial Neural Networks and Their Use in Chemistry

Statistical tests that tell you how important a variable is in determining category membership may not be reliable if these assumptions are not valid. If the assumptions are valid, and you need an explanation, but your categories are not linearly separable, you may be able to use discriminant analysis by invoking the trick illustrated in Figure 3c. We have simply created a new category from part of an old one (1), resulting in three categories that are roughly linearly separable.

Most statistical clustering methods are memory intensive and are simply unwieldy if the data set is too large. Also, some methods rely on assumptions (normal distribution, etc.) about data in forming clusters. So, if your data set is large or if it does not meet the necessary assumptions you may be better off using an ANN.

Modeling

Modeling problems are also encountered frequently by chemists. Optimization, signal processing, resolution, calibration, parameter estimation, and quantitative structure–activity relationships (QSAR) all require the modeling of a physical system. Optimization is concerned with the minimization or maximization of a function of one or more independent variables. The function need not be known a priori. Usually, one is interested in the values of the independent variables at the maximum or minimum values of the function. Signal processing is concerned with the development or use of methods for enhancing measurements with respect to chemically or physically relevant information. These methods include time series analysis, digital filtering for signal enhancement, smoothing, deconvolution, background correction, and image analysis. Peak detection and thereby the recovery of pure component spectra from the overlapped spectra of mixtures is the domain of resolution. Calibration is concerned with relating, correlating, or modeling a measured response based on amounts, concentrations, or other chemical or physical properties of a system, while parameter estimation entails the modeling of the properties of a system.

Although QSAR is very important, we will have very little to say about it, because the applications of ANNs to it are the subject of a recent book.21 Most of these application areas have a well-developed set of "traditional" (i.e., non-ANN) methodologies associated with them. It would lead us too far astray to discuss the pros and cons of each one relative to ANNs. We will simply state, as before, that generally the more nonlinear a data set is, the more likely that ANNs can be used to advantage. However, in comparison with the traditional methods, it will usually be more difficult to analyze an ANN to determine how it makes its decisions.

Using a neural network to model a system always entails supervised learning, and much of what we said about such learning in regard to classification problems will hold here as well. The largest difference between classification and modeling is that in the latter the output PEs represent continuous instead of discrete variables. Input/output pairs are typically referred to as variable-response pairs. Perhaps the simplest example of modeling is the use of Beer's...
What Can Neural Networks Be Used for?

A = abc. Recall the use of Beer's law to form a calibration curve: measure A for each of, say, five solutions having different concentrations of some absorber. Plot A versus c and find the least-squares line, \( A = mc + d \), that best describes the data. This yields two parameters, \( m \) and \( d \), that characterize the model and allow the concentration of a solution to be calculated from its absorbance.

Now consider a neural network approach. If you are an analytical chemist, you are interested in obtaining \( c \) from a measurement of \( A \). That is, the independent variable is \( A \) and the response is \( c \). An ANN would have one input PE corresponding to \( A \), a to-be-determined number of HL PEs and one output PE corresponding to \( c \). If you are an instrument manufacturer, though, you may be interested in \( A \) as a response to \( c \), in which case the input PE would correspond to \( c \) and the output PE to \( A \).

ANNs are typically not used to model Beer's law for a one-component system, but the example does illustrate most of the salient points of modeling. The variables are continuous; small changes in inputs cause small changes in outputs. Your point of view can determine what to consider a variable and a response. The choice of a response is not necessarily dictated by a theoretical equation written in its customary form (\( A = abc \) suggests that the dependent variable \( A \) is the response, but to the analytical chemist, \( c \) is usually the response). There may be an equation on which a "classical" model is based, but the ANN does not need to know it because it learns from example. The ANN can model a response without any a priori knowledge. This is very useful when no theoretical model exists. If there is a theoretical model, you may be better off using it, because the parameters in it will often have a physical interpretation and give an indication of what variables have the greatest effect on the response. (Again, this information may be obtainable from an ANN, albeit with more difficulty; see the last section of this chapter.) Of course, most parameters are obtained from least-squares fits which give rise to statistical tests of significance for the parameters. As usual, for these tests to be meaningful, your data must satisfy certain assumptions.

Some modeling problems can be formulated to involve several variables associated with one response, and others will involve several variables associated with several responses. ANNs typically have an advantage in the latter situation (but see the later section on Practical Considerations) because a single network can learn all the associations between the inputs and the outputs. In conventional modeling, you would simultaneously solve a set of equations. This set would have one equation for each response, and each equation would be a function of some combination of the input variables and some parameters. This set of equations may be underdetermined (no solution for all the parameters), or overdetermined (more equations than parameters), which can cause problems as well.

**Mapping and Associations**

Although anything an ANN does may be considered to be a mapping from input to output variables, here we have a specific meaning in mind. We
have seen one application/interpretation of mapping—the clustering of data. The other main application of mapping is the reduction of the dimensionality of a data set. Suppose that we have a data set with five input variables—this is a five-dimensional (input) data set—and for some reason we need or want to consider only two variables. Perhaps some of the variables are correlated; perhaps it takes too long to train a network; or perhaps we need two variables to more easily visualize the data. How can we accomplish this reduction of dimensionality? We can set up an autoassociative ANN (perhaps a backpropagation network) with five input PEs, five output PEs, and two HL PEs, and train it with supervision. If this ANN can be trained satisfactorily, it will contain a two-dimensional representation of the original data. The data will have been compressed. This is nice, but since the output is five-dimensional, how do we obtain the two-dimensional data? After training, pass the first training vector through the network. The outputs of the two HL PEs comprise the two-dimensional representation of this training vector. By passing each training vector through the network and recording the outputs of each HL PE for each vector, we obtain the "reduced" data set. The reduced data set could then be used as input to another ANN, or it could plotted to provide a visual representation of the data. Detailed examples of data reduction are given elsewhere.\textsuperscript{53,54}

Data compression is also useful for very large data sets. Suppose we have a spectral library containing 10,000 spectra that we wish to store. To obtain sufficient resolution, each spectrum may be digitized into 500 or even 1000 segments. This amounts to between 5 and 10 million pieces of data. An ANN with 500 input PEs, 500 output PEs, and a to-be-determined number of HL PEs can be trained. The outputs of the HL PEs constitute the reduced or compressed spectral data set.

Two important questions must be answered in these applications: (1) What do we mean by satisfactory training? and (2) How many HL PEs are required? Often these questions are interrelated. If we want a convenient means of visualizing data, then the obvious answer to the second question is to use two or perhaps three HL PEs. If we are trying to accomplish a more general data reduction, then the number of HL PEs must be sufficient to enable the network to learn outputs that are satisfactorily close to the inputs. This begs the question of what "satisfactory" means, but it is difficult to give meaningful guidelines in this extremely application-dependent matter. In the case of spectral compression, "satisfactory" means that the input and output spectra match closely enough to suit your purposes. Ultimately, you are the only person who can make this decision.

PCA is a traditional statistical method often used for reducing the dimensionality of data. The drawback of this method is that it is essentially linear and may therefore remove important nonlinearities in a data set. It is especially important to realize this when you are contemplating the use of PCA as a preprocessing step for input data to an ANN. Neural networks "thrive" on nonlinear data and in using principal components it is likely that you are
discarding some amount of useful information. It is worth noting in passing that ANNs have been used to perform PCA.\textsuperscript{55–57}

Multidimensional scaling is a nonlinear statistical mapping technique that can be used in data reduction and compression. Its use by the chemical community appears to be virtually nonexistent, and we merely refer the reader to the text by Schiffman et al.\textsuperscript{5}

As with mapping, anything an ANN does may be considered to be an association. We have seen examples of ANNs operating in both heteroassociative and autoassociative modes. Here we discuss one other application of an autoassociative ANN. It should be apparent by now that the weights of an ANN in some sense constitute the memory of the ANN. That is, the weights of the ANN are storing some representation of the data used to train the ANN. Suppose we have the digitized spectral library mentioned above and simply wish to store it in an ANN. We can set up an autoassociative ANN (perhaps a Hopfield network,\textsuperscript{50} although to store this many spectra would require a huge network) and train it with the library. When trained sufficiently, the ANN will have stored the library. You may think “So what? Why not just store the library on a hard disk and dispense with a training procedure and an ANN?” Suppose we are going to use our stored library to identify the spectrum of some compound. We would search the library until we found the spectrum that matched the given spectrum. This could also be done without an ANN. Unfortunately, most spectra in libraries tend to have been recorded under a set of prescribed conditions considered to be nearly ideal by the spectroscopist. The given spectrum probably was not recorded under the same conditions, and worse, these conditions probably were not ideal. The given spectrum is probably corrupted in some way relative to the library spectra. This will make finding a match more difficult and will require you to have some criteria for what constitutes a match. Statistical methods for library searching exist,\textsuperscript{58} but certain types of ANN are particularly adept at associating corrupted data. Our given spectrum may be corrupted with noise or with a drifting baseline, but if we were to pass it through the trained ANN, the output of the ANN would be the spectrum in its memory that most closely matches the given spectrum. See Refs. 59 and 60 for detailed examples of such methods.

**General Comments on ANNs, Statistics, and Artificial Intelligence**

In this section, we outline various types of data analysis in which ANNs are generally useful. We provide a general philosophy to guide you in when to use ANNs and when to consider statistical methods, and to alert you to the kinds of problem that can arise. The remaining sections of this chapter are quite detailed in discussing how ANNs operate, how to implement them, how to evaluate their performance, and what to do when things go wrong. A comprehensive discussion of statistical methods is beyond the scope of this tutorial,
but you are strongly urged to become familiar with the techniques that are applicable to your particular data analysis problems. Nonlinear statistical techniques are becoming more prevalent and may compete favorably with ANNs for some types of data. ANNs are not necessarily the panacea to all your analysis problems!

We have not yet mentioned artificial intelligence (AI) or expert systems (ES), and we end this section with a brief comparison between the latter and ANNs. An ES is a system that describes the behavior of an expert in some field by compiling the knowledge of the expert in the form of rules. This sounds like an ideal way to solve a problem, but there are several possible pitfalls. For more details, consult Ref. 61. Expertise may be due to chance or intuition. Expertise may not be expressible. Assuming that expertise can be expressed in language or symbols, it may not be understandable or applicable. Finally, expertise may be irrelevant, incomplete, or possibly even incorrect. These are all difficulties in acquiring the necessary knowledge base. In addition, ESs are brittle; if you give them incorrect or noisy data, you are likely to get the wrong answer. And, ESs are unable to generalize; they know what they know, but if they are confronted with a new situation they are likely to give the wrong answer.

ANNs learn by example and thereby generate their own rules. This alleviates the problem of acquiring and expressing a set of rules. The downside to this is that an ANN must be presented with enough examples in the training set to enable it to generate a sufficient set of rules. If you do not have enough examples, an ES or statistical method may be better suited to your problem. ANNs have a distributed memory—information is stored in all of the weights throughout the network. This distribution affords the ability to generalize effectively in the presence of new or noisy data. It also makes ANNs fault tolerant, which means that if some PEs are destroyed or disabled, or have their connections slightly altered, the behavior of the network is only slightly impaired.

You should be aware that ANNs and ESs can be combined into expert networks. This often requires the use of fuzzy arithmetic and logic (see Chapter 9 of Ref. 62 for an introduction). So when should you use an ANN as opposed to an ES or statistical method? Only you can provide the answer. We have presented a variety of issues to consider when making a choice. However, you are most familiar with your data and therefore best equipped to consider the issues involved in deciding. In the final analysis, if you are looking for the best answers to your problems, you may need to try several methods to see which one in fact performs best for you.

**PROCESSING ELEMENTS**

To understand how any particular type of neural network operates, we need to first consider how an individual PE functions. A PE can perform seven operations: summation, transformation, scaling and limiting, competition and/
or output, computation of error, backpropagation of information, and learning or adaptation. Depending on the type of network, a PE may or may not perform all seven of these operations. The order above describes the flow of information into, through, and out of the PE. Each operation is carried out by using a particular function. After a general description of PE operation, these functions are presented in detail.

A PE has one or more inputs. If the PE is not in the input layer, these inputs are the outputs of other PEs and will have weights associated with them. The first step in PE operation is to sum the inputs with a summation function. The result of the summation function may be thought of as the effective input to the PE. This effective input is then transformed via a transfer function, which depends on the effective input and an arbitrary but adjustable parameter typically referred to as “gain.” (In some instances, noise may be added to the effective input to a PE before the transfer function is applied. Typically a random number within a specified range is added to the effective input to each PE within a layer. The distribution of random numbers is either uniform or Gaussian.) The result of transformation, $T$, is then scaled linearly according to

$$MT + B$$  \[1\]

where $M$ and $B$ are constants: $M$ is referred to as the scale factor, and $B$ as the offset. The result of Eq. [1] may be hard-limited to upper and lower bounds. For example, if the upper bound is 5000 and the lower bound is $-5000$, then any number between and including these limits is left unchanged, but a number such as $-7500$ will be changed to $-5000$.

Next, the PE outputs a number through an output function, which depends on the value of the transfer function. In some types of network the PE will compete with other PEs in the same layer. The winner (or winners) of the competition outputs a value determined by the output function. The losers output a value of 0.0. PEs within a layer can also compete to see which one(s) will participate in learning or adaptation. This is discussed below. Next, an error may be computed. This error, which is the difference between the actual and desired outputs of the PE, may be subsequently transformed with an error function. Error computation is most common for PEs in the output layer of supervised learning networks. It is inherent to the operation of some network types, most notably backpropagation. However, many software packages will compute the error regardless of whether it is inherent to a specific network type, most often as a convenient means (though perhaps not the best; see the later section on Practical Considerations) of letting the user determine when to stop training.

After error computation and transformation, a PE may pass information to PEs in a previous layer. This information is either the transformed error, the transformed error scaled by the derivative of the transfer function, or the desired output. Finally, the weights of a PE are modified according to a learning rule or learning function. (Recall that these weights are those associated with
the inputs to the PE.) If some form of competition for learning is in effect, then only the weights belonging to the winning PE(s) are updated. There are many different and widely differing learning rules, but they typically depend on one or more possibly adjustable constants (learning coefficients), weights prior to an update, and one or more of the following properties of a PE: effective input, output, desired output, error, raw inputs, and the last change or update of weights.

We now enumerate some of the more common functions briefly noted above. The lists that follow are not comprehensive, but are highly representative. Some software packages may incorporate slightly different versions of the equations presented. Portions of the remainder of this section are adapted from Ref. 63.

**Summation Functions**

The following notation is used:

\[ i \] The current PE.
\[ j \] The PE from which \( i \) is connected.
\[ W_{ij} \] The weight in the connection that goes from \( j \) to \( i \).
\[ X_j \] The output of \( j \).
\[ I_i \] The result of the summation function for \( i \). This value is the effective input to \( i \) and will be subsequently transformed via a transfer function.

**Definitions**

**Weighted Sum** This is by far the most common summation function. The output of each PE \((j)\) connecting to \( i \) is multiplied by the weight in the connection from \( j \) to \( i \):

\[ I_i = \sum_j W_{ij} X_j \]  \[2\]

**Cumulative Sum** The weighted sum is added to the old weighted sum \( I_{i,\text{old}} \):

\[ I_i = I_{i,\text{old}} + \sum_j W_{ij} X_j \]  \[3\]

**Maximum** The maximum of the terms in the weighted sum (Eq. [2]):

\[ I_i = \text{MAX}_j(W_{ij} X_j) \]  \[4\]
**Minimum**  The minimum of the terms in the weighted sum (Eq. [2]):

\[ I_i = \text{MIN}_j(W_{ij}X_j) \]  

**Majority**  The number of terms in the weighted sum that are greater than zero minus the number of terms in the weighted sum which are less than or equal to zero:

\[ I_i = \sum_j \text{sgn}(W_{ij}X_j) \]  

where \( \text{sgn}(x) = 1 \) if \( x > 0 \), and \( \text{sgn}(x) = -1 \) if \( x \leq 0 \).

**Weighted Product**  Similar to a weighted sum:

\[ I_i = \prod_j W_{ij}X_j \]  

**Euclidean**  The Euclidean distance between the weight vector \( W_i = (W_{i,1}, \ldots, W_{i,n}) \) and the output vector \( X = (X_1, \ldots, X_n) \), where \( n \) is the number of connections to the \( i \)th PE.

\[ I_i = \left[ \sum_j (X_j - W_{ij})^2 \right]^{1/2} \]

**City Block**  The city block distance between the output vector and the weight vector. It is defined as follows:

\[ I_i = \sum_j |X_j - W_{ij}| \]

**Transfer Functions**

The following notation is used.

- \( I_i \): The result of the summation function for \( i \).
- \( T_i \): The result of the transfer function for \( i \) after it acts on \( I_i \).
- \( G \): Gain (a scalar number).

**Definitions**

*Linear or Direct*

\[ T_i = I_i \]
Artificial Neural Networks and Their Use in Chemistry

Sigmoid Maps $I_i$ into the range 0.0 to 1.0:

$$T_i = \frac{1}{1 + \exp(-I_iG)}$$  \[11\]

Hyperbolic Tangent Similar to the sigmoid function except that it maps $I_i$ into the range -1.0 to +1.0:

$$I'_i = I_iG$$  \[12\]

$$T_i = \frac{\exp(I'_i) - \exp(-I'_i)}{\exp(I'_i) + \exp(-I'_i)}$$  \[13\]

Sine:

$$T_i = \sin(I_iG)$$  \[14\]

Brain-State-in-a-Box (BSB) Similar to the linear function except that $I_i$ is scaled by $G$:

$$T_i = I_iG$$  \[15\]

Perceptron A piecewise linear function that is either zero or positive:

$$T_i = I_i \quad \text{if } I_i > 0$$
$$= 0 \quad \text{if } I_i \leq 0$$  \[16\]

Sometimes however, if $I_i > 0$, then $T_i$ is set to 1, and if $I_i \leq 0$, then $T_i$ is set to zero. In this case, the PE is said to be a binary or threshold logic unit.

Sign Equal to +1 if $I_i > 0$. Otherwise it is -1.

$$T_i = 1 \quad \text{if } I_i > 0$$
$$= -1 \quad \text{if } I_i \leq 0$$  \[17\]

Modified Sign Similar to the sign function, except that if $I_i = 0$, then $T_i = 0$. That is,

$$T_i = 1 \quad \text{if } I_i > 0$$
$$= 0 \quad \text{if } I_i = 0$$
$$= -1 \quad \text{if } I_i < 0$$  \[18\]
**Step Function**  Similar to the sign function except the smallest value is 0, instead of -1. Thus, it is a binary logic unit.

\[
T_i = \begin{cases} 
1 & \text{if } I_i > 0 \\
0 & \text{if } I_i \leq 0 
\end{cases} \tag{19}
\]

**Exponential:**

\[
T_i = \exp(I_i) \tag{20}
\]

**Bidirectional Associative Memory (BAM)** Similar to the modified sign, except in the way it treats the \( I_i = 0 \) case:

\[
T_i = \begin{cases} 
1 & \text{if } I_i > 0 \\
T_i & \text{if } I_i = 0 \\
-1 & \text{if } I_i < 0 
\end{cases} \tag{21}
\]

**Output Functions**

The following notation is used:

- \( T_i \) The result of the transfer function for \( i \) after it acts on \( I_i \).
- \( O_i \) The output value of \( i \) after the output function acts on \( T_i \).
- \( T \) The vector of transfer values for a layer of PEs. For a layer with \( N \) PEs, \( T = (T_1, T_2, \ldots, T_N) \).

**Definitions**

**Normalized Output** The output of \( i \) is normalized by the length of the \( T \) vector, \( \|T\| \):

\[
O_i = \frac{T_i}{\|T\|} \tag{22}
\]

**Soft or Softmax:**

\[
O_i = \frac{\exp(T_i)}{\sum_j \exp(T_j)} \tag{23}
\]

**Linear Direct:**

\[
O_i = T_i \tag{24}
\]

**Competitive Direct** Similar to direct, except that it depends on whether a PE has ever learned. This is used in conjunction with learning rules that are
Artificial Neural Networks and Their Use in Chemistry

competitive, as explained in detail after Eq. [25]. If the learning rule is not competitive, then whether \( i \) has learned is irrelevant, and the direct output function, Eq. [24] applies.

\[
\begin{align*}
O_i &= T_i & \text{if } i \text{ has learned} \\
&= 0 & \text{if } i \text{ has never learned}
\end{align*}
\]  

[25]

In addition to this equation, the PE with the greatest transfer value in the layer is chosen for learning. If this chosen PE (the \( i \)th PE) has never learned, then Eq. [25] is ignored and \( O_i = T_i \). Taken literally, Eq. [25] and the preceding two sentences imply that there can be only one winning PE in a layer, that this PE is the only PE that learns or adapts, and that all other PEs in the layer will not learn or adapt. Some software packages will let the user specify more than one winner in a layer. In this case, two PEs, for example, may win and adapt, while all others in the layer will not. The output function will work similarly to Eq. [25], except that typically the output for both PEs will be the average of the transfer values for each PE.

We have used the phrase “winning PE” rather glibly, without explaining what it means. Usually the PE with the largest value of \( T_i \) in a layer is the winning PE. However, if the Euclidean distance or city block distance summation function is used, the opposite holds true, that is, the PE with the smallest value of \( T_i \) is the winner. These summation functions are usually used with special-purpose transfer functions (an example is a radial basis transfer function), which perform a monotonic inversion of the effective input \( I_i \).

Error Functions

Definitions

*Standard* No error transformation.

*Quadratic* Error is squared, but usually retains its sign.

*Cubic* Error is cubed.

Learning Rules

The following notation is used:

\[ C_1, C_2, C_3 \]

These constants (perhaps adjustable over the course of training) are learning rates, learning coefficients, thresholds, etc., depending on the learning rule. The role each plays will be identified in the following learning rules.

\[ I_i \]

The result of the summation function for \( i \).

\[ O_i \]

The output value of \( i \).
\( D_i \) The desired output of \( i \) (i.e., what the output of \( i \) should be).

\( E_i \) The error of \( i \). If \( i \) is in an output layer, then \( E_i \) is either the transformed error or the transformed error scaled by the derivative of the transfer function. If \( i \) is not in an output layer, then \( E_i \) is the transformed back-propagated error.

\( X_{ij} \) The input from \( j \) to \( i \).

\( W_{ij} \) The weight in the connection that goes from \( j \) to \( i \).

\( W'_{ij} \) The weight after being updated by the learning rule in the connection that goes from \( j \) to \( i \).

\( M_{ij} \) The value of the last change in the weight \( W_{ij} \).

\( A_{ij} \) An auxiliary weight.

\( n \) The number of inputs to \( i \).

\([x]^{(1)}\) A function defined as follows:

\[
[x]^{(1)} = \begin{cases} 
1 & \text{if } x > C_2 \\
0 & \text{if } x \leq C_2
\end{cases}
\]  

\([x]^{(p)}\) A function defined as follows:

\[
[x]^{(p)} = \begin{cases} 
1 & \text{if } x > 0 \\
0 & \text{if } x \leq 0
\end{cases}
\]  

**Definitions**

*No Learning Rule* Weights for \( i \) do not change.

*Hebb* If the desired output of \( i \) and input to \( i \) are greater than some threshold value \( C_2 \), then the weight is changed by the learning rate \( C_1 \). Mathematically, this is accomplished by

\[
W'_{ij} = W_{ij} + C_1[D_i]^{(1)}[X_{ij}]^{(1)}
\]  

Note that application of Eq. [26] gives

\([D_i]^{(1)} = 1 \text{ if } D_i > C_2 \quad \text{and} \quad [X_{ij}]^{(1)} = 1 \text{ if } X_{ij} > C_2\]

That is, if \( D_i \) and \( X_{ij} \) are greater than the threshold \( C_2 \), the weight \( W_{ij} \) is changed by \((C_1)(1)(1) = C_1\), the learning rate.

*Hebb/Anti-Hebb* If the desired output of \( i \) and the input to \( i \) are greater than the threshold \( C_2 \), this is the same as the Hebb rule, Eq. [28]. If the desired output of \( i \) is greater than the threshold, but the input to \( i \) is less than the
threshold, then the weight is decreased by the learning rate \( C_1 \); this is anti-Hebb learning. Hebb and anti-Hebb learning can be expressed mathematically as one equation:

\[
W'_{ij} = W_{ij} + C_1 [D_i]^{(1)} [2X_{ij}]^{(1)} - 1
\]

\[29\]

**Hopfield** If the desired output and the input are both greater than threshold or both equal to or less than threshold, the weight is changed by the learning rate. If neither of these conditions holds, then the weight is decreased by the learning rate:

\[
W'_{ij} = W_{ij} + C_1 [2[D_i]^{(1)} - 1][2X_{ij}]^{(1)} - 1
\]

\[30\]

The three learning rules above are used in a Hopfield network, with \( C_1 \) typically equal to 1.

**Perceptron** If the actual output of \( i \) is active (greater than threshold, which is zero for this case) and should be active, or if the output of \( i \) is inactive (equal to or less than threshold) and should be inactive, then there is no learning for \( i \) (\( W_{ij} \) is unchanged). If the actual output should be active but is not, then the weights on active input connections are increased. If the actual output should be inactive but is not, then the weights on active input connections are decreased. Mathematically, an input connection is active if the input is above threshold (i.e., zero):

\[
\begin{align*}
W'_{ij} &= W_{ij} \quad \text{if } [2[D_i]^{(p)} - 1][2[O_i]^{(p)} - 1] > 0 \\
&= W_{ij} + k_i [2[D_i]^{(p)} - 1][X_{ij}]^{(p)} \quad \text{otherwise}
\end{align*}
\]

\[31\]

where

\[
k_i = (C_1(D_i - O_i)/n) + C_2
\]

\[32\]

This rule is used in the output layer of the perceptron network; \( C_1 \) is usually 0.0, and \( C_2 \) is usually a small number such as 0.05. Note that as such, \( C_1 \) and \( C_2 \) do not play the roles of learning rate and threshold, respectively.

**Brain-State-in-a-Box (BSB) Hebb** This is sometimes referred to as Hebbian learning and is a bipolar (i.e., input and output values are restricted to +1 or −1) version of the Hopfield rule. If the desired output of \( i \) and the input to \( i \) are both +1 or both −1, then the weight is increased. If either the desired output of \( i \) or the input to \( i \) is zero, then there is no learning. For all other cases, the weight is decreased. Mathematically, we write:

\[
\begin{align*}
W'_{ij} &= W_{ij} + C_1 D_i X_{ij} + C_2 M_{ij} \\
M'_{ij} &= W'_{ij} - W_{ij}
\end{align*}
\]

\[33\]
This rule is used in the middle two layers of the biassociative memory (BAM) network and also in the Hebb version of the BSB network. $C_1$ is the learning rate and usually is set to 1. $C_2$, the momentum, or momentum coefficient, is usually not used (i.e., $C_2 = 0$).

**BSB Widrow-Hoff** The weight $W_{ij}$ is changed in proportion to the error in the output, $D_i - O_i$. Each change reduces the error.

$$W_{ij}' = W_{ij} + C_1(D_i - O_i)X_{ij} + C_2M_{ij}$$

$$M_{ij}' = W_{ij}' - W_{ij} \tag{34}$$

This rule can be used in a Hopfield network, in the middle layer of the BSB network, and in the outer layer of a counterpropagation network. In the latter case, it is equivalent to the so-called Grossberg outstar learning rule. $C_1$ is usually set to 0.1 or less and $C_2$ is usually set to zero.

**Adaline** Weights are changed so that the error is reduced by $1/n$, where $n$ is the number of inputs to $i$, thereby reducing the difference between the weighted sum $I_i$ and the desired output $D_i$ to zero:

$$W_{ij}' = W_{ij} + \frac{C_1}{n} (D_i - I_i)X_{ij} \tag{35}$$

This rule is used in the adaline layer of the adaline and madaline networks, for which the only allowed values for $X_{ij}$ are +1 and -1. Weights can change even if the output is correct.

In the madaline network, the PEs in the adaline layer compete for learning, the winner being the PE whose weighted sum is closest to zero but with the wrong output. Only the winning PE learns. The learning rate $C_1$ is usually set to 1.

**Kohonen** The weights are changed so that the weight vector for the layer containing $i$ [i.e., the vector $(W_{i1}, W_{i2}, \ldots, W_{in})$] more nearly approximates the input vector [i.e., the vector $(X_{i1}, X_{i2}, \ldots, X_{in})$] which caused $i$ to win. If $i$ has not learned before, then $C_j$ is set to 1 for $i$. Otherwise $C_1$ is usually some small number, say 0.1. The individual weights in the weight vector learn according to:

$$W_{ij}' = W_{ij} + C_1(X_{ij} - W_{ij}) \tag{36}$$

**Delta Rule** The weights are changed similarly to the Widrow-Hoff rule, Eq. [34]. Mathematically, we have:

$$W_{ij}' = W_{ij} + C_1E_iX_{ij} + C_2M_{ij}$$

$$M_{ij}' = W_{ij}' - W_{ij} \tag{37}$$
In Eq. [37], \(E_i\) is the error of the output layer after it has been scaled by the derivative of the transfer function of the output layer, \(C_1\) is the learning rate, and \(C_2\) is the momentum, which may or may not be present. The difference between Eqs. [34] and [37] is that \(D_i - O_i\) has been replaced by \(E_i\) in order to indicate that the error for the delta rule is calculated differently from the error for the Widrow-Hoff rule. The error for a PE in the output layer of a network (this is not necessarily the layer the current PE resides in—although it may be—but, rather, the final or last layer of the network) is calculated as the difference between the desired output and the actual output for that PE. This error is scaled by the derivative of the transfer function for the output layer and passed backward to the layer or layers preceding the output layer. The error for a PE in this preceding layer (the PE of interest) is the error scaled by the derivative of the transfer function. That is, \(E_i\) in Eq. [37] is the error of the output layer after it has been scaled by the derivative of the transfer function of the output layer. Usually \(C_1\) is set between about 0.1 and 0.5 during the early phases of training and decreased as training progresses. The learning rate for a network output layer with a linear transfer function should be about an order of magnitude smaller. Generally, learning rates should be largest for the first HL and progressively decrease toward the output layer. When \(C_2\) is present, it tends to smooth out weight changes. Training data should be arranged randomly to prevent oscillatory changes in weights and possible nonconvergence.

**Cumulative Delta Rule** This is a variant on the delta rule. In the delta rule, the weights are updated after the presentation of each input/output pair of data. In the cumulative delta rule, the weight changes are accumulated and then applied all at one, typically at the end of an integer multiple of the training epoch. Mathematically, we write:

\[
M'_{ij} = M_{ij} + C_1 E_i X_{ij} \quad \text{after presentation of each input/output pair} \quad [38]
\]

\[
W'_{ij} = W_{ij} + M_{ij} + C_2 a'_{ij} \quad \text{after presentation of integer multiple of training epochs} \quad [39]
\]

\[
a_{ij} = M_{ij}
\]

\[
M'_{ij} = 0
\]

This rule is used in an attempt to overcome a structured presentation (i.e., nonrandom ordering of cases) of a training set. The larger the integer multiple of epochs, the smaller \(C_1\) should be to prevent weight changes from becoming too large, which would cause learning to diverge. Comments about \(C_1\) for the delta rule also apply here. The normalized cumulative delta rule automatically decreases the learning rate \(C_1\) as a function of the number of epochs of training that has transpired.
Before presenting descriptions of specific ANNs (in the next section), we discuss in general terms the various architectures and dynamics of networks. The architecture of an ANN refers to the physical organization and arrangement of the PEs in the network and also how the PEs are connected. These considerations are important because they help bring order to the many different types of ANN that exist by placing them into categories.

Groups of PEs can be characterized by the number of layers they form, the number of PEs in each layer, and the manner in which the PEs are connected. Connections between layers may be in the forward or backward direction, whereas connections within a layer may be either inter-PE (i.e., lateral), or intra-PE (i.e., a PE connected to itself). We may also speak of the degree of connectivity. Figure 4a shows two layers which are fully interconnected; each PE in one layer is connected to every PE in the other layer. Figure 4b shows two layers that are correspondingly connected; each PE in one layer is connected only to the PE above (below) it. Figure 4c shows two layers that are round-robin connected; the first, third, fifth, . . . , PEs in the bottom layer are connected to the first PE in the top layer, and the second, fourth, sixth, etc. PEs in the bottom layer are connected to the second PE in the top layer. If the top layer were to contain four PEs, then the first, fifth, etc. bottom-layer PEs would be connected to the first top-layer PE, etc. Figure 4d shows a layer fully connected to itself. Finally, in some networks PEs in one layer may be connected to randomly selected PEs in another layer. Full, corresponding, round-robin, and random connections may
be forward and/or backward. Connections shown in Figure 4 are forward; appropriate arrows would indicate other directions. Layers connected to themselves are usually fully connected.

Based on the foregoing architectural considerations, we find that most ANNs fall into one of six categories. (Note: We have not included input or output layers in distinguishing between these categories of networks. Rather, we have indicated only the number of hidden layers: e.g., the single-layer, laterally connected ANN would typically have an input and an output layer in addition to one laterally connected hidden layer.) This is not the only possible classification scheme. Under certain conditions it may be possible to place a given network into more than one category.

Multilayer, feedforward ANNs are adept at solving classification problems and when properly trained are often capable of generalization. Signals pass through the network in a forward direction only. Layers may be fully, correspondingly, round-robin, or randomly connected. The perceptron, adaline, madaline, and backpropagation networks are examples of this, as are the Boltzmann and Cauchy machines. (In this section we give references only for networks that are not discussed in the next section.) The word “backpropagation” refers to the use of forward signals to change weights of PEs in a previous layer (see previous section on Processing Elements) and not to the direction of the signals themselves.

Single-layer, laterally connected ANNs are autoassociative. They can store many data vectors and are adept at outputting one of these vectors when presented with a noisy or incomplete version of it. Examples are the Hopfield, brain-state-in-a-box, and sparse distributed memory networks.

Single-layer, topologically ordered ANNs may be autoassociative (learning vector quantization network) and therefore are useful in data compression and preprocessing data for classification purposes. The topology-preserving map or Kohonen network is useful for mappings (clustering and dimension reduction) and optimization problems. We explain topology in more detail in the next section.

Two-layer feedforward/feedback ANNs are heteroassociative. They can store input and output vectors and are useful in recalling an output vector when presented with a noisy or incomplete version of its corresponding input vector. They are also useful for classification problems. Typically, every feedforward connection between two PEs is accompanied by a feedback connection between the same two PEs. Both connections have weights, and these weights are usually different from each other. Examples are the adaptive resonance theory and bidirectional associative memory networks.

Multilayer, cooperative/competitive ANNs are networks containing several layers, at least one of which is laterally (usually fully) connected. Layers may have forward and backward connections. The lateral connections allow PEs within a layer to either compete (see previous section on Processing Elements) or cooperate. The weight of a competitive connection is negative and
said to be inhibitory, whereas the weight of a cooperative connection is positive and said to be excitatory. These networks are capable of performing very complicated mappings and are useful in classification and clustering problems. Examples are the Rumelhart–Zipser, masking field, boundary contour system, hierarchical scene structure, and neocognitron networks.

Hybrid networks combine the features of two or more types of ANN, the idea being to highlight the strengths and minimize the weaknesses of the different networks. Examples are the Hamming network, which has a perceptron-like and a Hopfield-like layer, and the counterpropagation network, which has a Kohonen layer and a Grossberg outstar layer.

It is also possible to construct a system of interacting ANNs; this can be useful if no known architecture is adequate for a complex problem. For an introduction to this option, refer to Chapter 13 of Maren et al.

The dynamics of an ANN specify what the network does during its operation. Some networks use different dynamics, depending on whether they are operating in training or test mode. The dynamics and architecture of an ANN are intimately linked; you would not expect a feedforward network to operate in the same fashion as a laterally connected or a feedforward/feedback network, primarily because the connection schemes are different for each. We did not say it at the time but our discussion in the Introduction of information flow through the network in Figure 2 was a simplified version of the dynamics of that network. That discussion specified the path of information flow and its timing. To flesh out the dynamics, we should specify the summation, transfer, output, and learning functions for each PE. Usually all PEs in a layer have the same path, timing, and functions. That is, the layer dynamics collectively form the global network dynamics.

A major issue of dynamics is that of network stability, which refers to one of two things: (1) Do the weights in the network converge to stable values? and (2) Is the network dynamically stable? The first question has to do with learning stability. A glance at the learning rules in Eq. [28]–[39] shows that they are written in terms of weight changes. A learning rule may be executed many times during a training procedure, but there is no guarantee that the weights will converge through this iterative process. If they do not converge, the network is not a learning-stable one, and you will see oscillations in agreement between input and output vectors as training proceeds. We will have more to say about this in the later section on Practical Considerations in Solving Problems with Neural Networks. The second question is more subtle and is applicable to networks having two-way connections between PEs. In a feedforward network dynamic stability is not an issue because information flows only from the input to output layers and then stops (for a given training pattern). In networks with two-way connections, however, information bounces between PEs, potentially forever. To prevent this, the network must converge to a stable solution. This occurs when the values stored in each PE remain constant, and this happens when the energy function associated with the ANN is minimized. Note that it is
possible for a dynamically stable solution to be wrong! It is beyond the scope of this chapter to further discuss energy functions and dynamic stability. Nevertheless, there is a great deal of theoretical research in these areas utilizing statistical mechanics and spin systems. For an introduction to this area, see Refs. 13 and 78–80.

DIFFERENT TYPES OF ARTIFICIAL NEURAL NETWORK

To provide you with a solid basis for deciding whether or not a given ANN is appropriate for your intended use, we describe briefly in this section many of the types of ANNs that have appeared in the literature in the past few years. For each network we focus on strengths and weaknesses, some practical aspects of operation, and a literature review of the chemical applications. We do not delve into detailed mathematical descriptions of networks, since these can be found in any number of texts. In particular we call attention to Ref. 19, which offers step-by-step developments of equations and detailed numerical examples for backpropagation, biassociative memory, counterpropagation, Hopfield, and Kohonen self-organizing map networks. Adaptive resonance theory networks are reviewed in detail in Ref. 27.

Adaptive Resonance Theory (ART) Networks

ART networks were developed by Grossberg and Carpenter as a solution to the "stability–plasticity dilemma," which refers to the difficulty of designing a network that is able to retain previously learned information while learning new information. The ART networks are two-layer feedforward/feedback heteroassociative networks that are most useful as pattern recognizers (classifiers) and learn in an unsupervised mode. Their ability to generalize is limited. ART1 networks are capable of processing only binary inputs, whereas ART2 networks are capable of analog inputs. ART architectures are relatively simple. Many adjustable parameters need to be set, however, and the theory behind their operation is quite complex. They have been characterized as "one of the most complex neural networks ever invented." References 27 and 81 supply more details on these networks.

Most uses of ART networks by the chemical community are very recent. In applications related to infrared spectroscopy, they have been used to recognize aromatic substitution patterns (they reportedly performed better than human experts) and also in the clustering of spectra of lubricating base oils. Reference 83 also contains a good comparison of ART networks with other methods. ART networks have also been applied in choosing a detector for ion...
Different Types of Artificial Neural Network

Recently, an ART network was used to simultaneously classify particles according to size and chemical composition from aerosol time-of-flight mass spectrometry (ATOFMS) data. ATOFMS can typically acquire data for 50–100 particles per minute, and the ART network was able to classify particles in less time than was required for data acquisition, thereby providing automated, online data analysis.

**Backpropagation (BP) and Related Networks**

Backpropagation networks are the best known, the most widely applied, and are the most thoroughly studied ANNs to date. They learn with supervision, accept continuous inputs and outputs, and may be either hetero- or auto-associative. The simple network diagrammed in Figure 5 is a generic BP network, which may be modified in several ways. There may be more than one HL. Summation functions for all layers are almost always weighted sums. Output functions for all layers are almost always direct. (The output layer may have a softmax output function to help the network handle one-of-N codes; see the later section on Practical Considerations.) The transfer function for the input layer is almost always linear, but for all other layers may be any differentiable function, the most common being the sigmoid, hyperbolic tangent, and sine functions. (Caution: The choice of transfer function can change the error quantity that is backpropagated through the network.) There is no learning rule for the input layer, but the other layers may use the delta rule or some variation of it, such as the cumulative delta rule. There is typically a bias PE that provides an input of one to every HL and output layer PE, but has an adaptable weight.

The operation of a BP network has been fully described elsewhere (see in particular Chapter 8 of Ref. 19), so we merely summarize its operation here. During the learning process, input variables flow toward the output layer, while

![Figure 5](image-url)  
*Figure 5* A simple backpropagation network. The label to the right of each layer gives the layer name, and the summation function, transfer function, output function, and learning rule, respectively, for the layer in a typical network. The input layer is fully connected to the middle layer, which in turn is fully connected to the output layer.
the output variables are applied to the output layer. (As usual, the number of input PEs is equal to the number of input variables, and the number of output PEs is equal to the number of output variables.) The actual output of the output PEs is compared to the desired output of these PEs, and the individual PE errors are "backpropagated" through the network (see the previous section on Processing Elements), where through the learning rules and a gradient descent method the weights are adapted to minimize the total output error. This error is a function of all weights in the network, and ideally the weights would change until the global minimum of this function was reached. In practice this seldom occurs; usually a local minimum is found instead. Often, a local minimum gives rise to satisfactory network performance if this minimum is deep enough. To prevent settling into a shallow local minimum, two (or more) adjustable parameters are used in the learning rule, the learning rate, and the momentum (see the later section on Practical Considerations in Solving Problems with Neural Networks). Other aspects of BP network operation are discussed throughout this chapter.

Backpropagation is a general-purpose ANN paradigm that can be used for modeling, classification, mapping, clustering, signal processing, and other functions. It is a good all-around network choice for many types of data analysis, but usually not the best choice. Its advantages are that it is a general non-linear regression technique that attempts to minimize global error, it can (in principle) synthesize any multidimensional function, and it can provide compact distributed representations of complex data sets.

Backpropagation networks have several disadvantages, which we will briefly discuss, indicating ways to overcome them. For example, BP networks tend to be slow learners; complex data sets may require many thousands of epochs. Also, it may be difficult to find values of the learning rate and momentum that promote fast learning. The Quickprop and Maxprop algorithms use quadratic estimation to determine the direction and step size on the error surface. In the fast learning algorithm, a portion of the estimated error is added to the output of a connection's source PE prior to changing the corresponding weight. The delta bar delta (DBD) method uses previous values of curvature of the error surface to infer the current curvature. This leads to a learning rule in which every weight has a different learning rate that is automatically calculated. The extended delta bar delta (EDBD) algorithm extends the DBD method by also calculating a momentum for each weight.

The highly distributed nature of the BP network is a disadvantage when one is trying to learn two or more fundamentally different problems (the output variables are likely to represent completely different types of quantities). In this situation the data is naturally "partitioned" into two or more regions or segments, but because of "spatial cross talk," a single network has difficulty separating and solving the problems. Either try to break the problem into two smaller ones or use a different network. For regression problems, appropriate choices are the generalized regression and radial basis function networks,
whereas classification problems are more likely to be amenable to learning vector quantization, probabilistic neural networks, or radial basis function networks. These last three networks are also useful if you suspect that a BP network is forming irregular decision boundaries between categories (as evidenced by unexpected classifications, i.e., incorrect classifications when the correct answer may seem obvious).

In many cases it is difficult to determine in advance how many hidden layers and how many HL PEs are required for satisfactory performance. A trial-and-error method to determine this information can be very time-consuming. Cascade correlation networks build HLs one PE at a time, solving a problem incrementally.87

BP networks tend to get stuck in local minima. Although the algorithms mentioned earlier that speed up learning help to overcome this tendency, it may be best to use a different network if you think this is a serious problem. The networks listed in the second to last paragraph for regression and classification problems are possible options.

We now briefly review chemical applications of BP networks. Many applications to infrared spectroscopy may be found in the literature. For example, in a calibration application, BP networks performed slightly better than partial least squares in the chemical composition analysis of carrageenans.91 Confidence intervals for the determination of aromaticity of coals from IR reflectance spectra have been calculated.92 Also, BP networks have generally been useful in the structural interpretation of IR spectra.93-102 However, one study found that, while useful, the performance was no better than a partial least-squares method, and both methods fell short of the results obtained by skilled human interpreters.94 The work by Klawun and Wilkins95 is an exhaustive BP study in the area of infrared spectroscopy which we consider to be exemplary in its thoroughness. An interesting approach to interpretation uses a hierarchy of BP networks to "zero in" on the presence or absence of structural features.97 BP networks have also exhibited large improvements in speed and accuracy in searching IR spectral libraries.98-100 An application to matrix isolation IR spectra led to the development of the flashcard algorithm, which overrepresents cases that are difficult to learn.101 A BP network was applied to automatic compound identification in low resolution, open-path Fourier transform IR spectrometry.102

Nuclear magnetic resonance (NMR) has also been a fruitful area of ANN BP application.103-111 Most studies have dealt with either the simulation of 13C spectra or the prediction of 13C shifts,103-105,107-109 although one study focused on prediction of phosphorus shifts.106 A BP network was used to predict secondary protein structure, which was then used to assist in NMR assignment.110 In another study, 1H NMR spectra of binary mixtures of alditols were successfully classified.111

BP networks were used for multivariate calibrations of pyrolysis mass spectra.112,113 In one case the BP networks gave better concentration predic-
tions than did both partial least-squares and principal components regression methods. Another study with similar results also noted that linear transfer functions gave better results than sigmoid functions. Structural features were successfully identified in a library search of mass spectra. Interlaboratory calibration of two mass spectrometers has been accomplished.

There exist a number of applications to a variety of other spectroscopic techniques. Fuels and oils were classified from laser-induced fluorescence spectra for which, interestingly, it was found that classification performance dropped when PCA was used to preprocess the data. Multivariate nonlinear calibration of fluorescence data was accomplished using a BP network with hidden node pruning. In the identification of Raman spectra it was found that, compared to other transfer functions, use of sine functions in the HL led to faster training and better discrimination between closely related spectra. Unfortunately, less tolerance to spectral distortions was also exhibited. BP networks were better than standard library searching methods for noisy UV spectra, provided a comprehensive training set was used. Signatures generated by laser desorption–ion mobility spectrometry were successfully classified by BP networks. As a final example of BP in spectral applications, we note that principal components were used as input to a BP network for the calibration of ion mobility spectrometry measurements.

A wide variety of other applications have been reported. High performance liquid chromatography (HPLC) mobile phase parameters have been optimized. Better predictions of HPLC capacity factors were obtained from BP networks than from multiple regression. The effect of five factors in ion interaction HPLC has been studied. BP networks were used to successfully predict optimal separation conditions in high performance capillary zone electrophoresis from a relatively small number of experiments. A comparison of the predictive power of BP networks with a theoretical equation (a “hard modeling” equation) in ion interaction chromatography showed that the two methods performed comparably as long as enough data was used; the smaller the amount of data, the worse the ANN performed. A BP ANN was found to be better in predicting gas chromatographic retention indices than multiple linear regression, which, in turn, was better than a counterpropagation ANN. Chiral chromatographic separations were predicted using a BP network.

BP processing of stripping analysis responses allowed the quantitative determination of heavy metals in the presence of interferences, and selectivity coefficients of berberine-selective electrodes were predicted satisfactorily. BP networks were used to analyze piezoelectric crystal data to simultaneously determine sulfur dioxide concentration and relative humidity. A BP network was used to model properties of materials. Two studies described the modeling and recognition of flow injection patterns. Compared to principal components regression and partial least squares (PLS), BP provided a better method for multicomponent kinetic determinations.
Different Types of Artificial Neural Networks

networks also successfully modeled a photocatalytic degradation.\textsuperscript{136} Stability constants of crown ethers have been predicted.\textsuperscript{137} Aqueous solubilities of heteroatom-containing organic compounds were predicted using a Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm (an optimization procedure)\textsuperscript{138} to determine network weights.\textsuperscript{139} A BP network was used to predict equilibrium bond lengths, dissociation energies, and fundamental frequencies of diatomic molecules.\textsuperscript{140} Modular BP networks were used to search for direct correlations between structure and properties of compounds.\textsuperscript{141} Invariant features of protein folding have been studied with BP networks,\textsuperscript{142} and a review of ANN and other new approaches to secondary and tertiary protein structure prediction has been published.\textsuperscript{143} A BFGS algorithm was used to predict supercritical carbon dioxide solubilities of organic compounds from their structures.\textsuperscript{144} Excellent prediction of atomic ionization potentials was achieved with a BP network.\textsuperscript{145} Potential energy surfaces were modeled using an adaptive, global, extended Kalman filter to determine weights.\textsuperscript{146} This method was shown to be better than MARS (a polynomial splines technique)\textsuperscript{146} and was reliable enough to significantly reduce the computational time and cost of calculating potential energies when compared with quantum mechanical methods. Potential energy surfaces were also successfully modeled in three other studies.\textsuperscript{147–149}

The BDB variant of BP was used to predict the density, viscosity, and refractive index of ternary and quaternary liquid systems based on training data from binary systems.\textsuperscript{150} Performance was acceptable, although a theoretical equation gave better predictions in a number of cases. BDB has also been applied to the prediction of bond dissociation energies.\textsuperscript{151}

The EDBD variant of BP was used in several chemical kinetic studies.\textsuperscript{152–154} This method gave much better estimates of kinetic analytical parameters than either nonlinear regression or principal components regression.\textsuperscript{152} EDBD has also been applied to multicomponent kinetic determinations\textsuperscript{153} and to the estimation of kinetic compartmental model parameters.\textsuperscript{154} The EDBD method was found to offer increased modeling power for nonlinear multivariate data compared to partial least squares and principal components regression, provided the training set is extensive enough to adequately sample the nonlinear features of the data.\textsuperscript{155} Finally, EDBD has been successfully applied to the prediction of retention indices.\textsuperscript{156}

Biassociative Memory (BAM) Networks

BAM networks are two-layer feedforward/feedback heteroassociative networks (they can also be autoassociative). An example network is shown in Figure 6. Standard BAMs take bipolar (±1) inputs and outputs. Adaptive BAMs (ABAM) can take continuous inputs and outputs. In either case, input data should be mutually orthogonal (i.e., independent, nonredundant, and uncorrelated). BAMs were inspired by ART networks but are conceptually sim-
output; weighted, BAM, direct, none

BAM 2; weighted, BAM, direct, BSB:Hebb

BAM 1; weighted, BAM, direct, BSB:Hebb

input; weighted, sign, direct, none

Figure 6 A simple biassociative memory (BAM) network; see Figure 5 for an explanation of the labels. The first BAM1 layer has the same number of PEs as the input layer and is correspondingly connected to it; the BAM2 layer has the same number of PEs as the output layer and is correspondingly connected to it. The two BAM layers are fully connected in both directions to each other.

pler. They can be thought of as generalizing Hopfield networks into hetero-associative ANNs. BAMs are good at storing input/output pairs, although their storage capability and generalization ability are somewhat limited. BAM networks differ from most other ANNs we have discussed in that they are trained in only one epoch; that is, one pass through the training set completely specifies the weights of the network. Alternatively, the training set is completely and exactly learned and stored!

The real utility of BAM networks is their ability, after training on input/output pairs, to associate corrupted inputs with their correct outputs (i.e., the output belonging to the uncorrupted input). They do this by iteratively changing their weights upon presentation of the corrupted input. This is usually referred to as learning. (Note that we have usually associated “learning” with “training” in this chapter. In a BAM network, learning—or the changing of weights—actually takes place during recall mode.) This learning is supervised and occurs by bidirectional flow of information between the BAM 1 and BAM 2 layers. Chapter 5 of Ref. 19 gives an explicitly detailed description of ABAM networks.

BAM networks have seen successful application to qualitative analysis of X-ray fluorescence\textsuperscript{157} and inductively coupled plasma–atomic emission spectra.\textsuperscript{157,158} They have also been applied to background prediction and correction of infrared spectra,\textsuperscript{159,160} giving better results than partial least squares.

**Counterpropagation Networks**

Heteroassociative networks termed “counterpropagation” can accept continuous inputs and outputs. They are two-layer ANNs (disregarding the
output; weighted, linear, direct, Widrow-Hoff

Kohonen; weighted, linear, competitive direct, Kohonen

input; weighted, linear, direct, none

Figure 7 A simple counterpropagation network; see Figure 5 for an explanation of the labels. The Kohonen layer is fully connected to the input layer, and the output layer is fully connected to the Kohonen layer.

input layer) consisting of a Kohonen layer and a Grossberg outstar layer, which doubles as the output layer. A simple counterpropagation network is shown in Figure 7. The input layer has one more PE than there are input variables. If a data vector is denoted as \( X = (X_1, X_2, \ldots, X_m) \) (there are \( m \) input variables), then the corresponding vector that is passed through the network is \( X' = (X_1, X_2, \ldots, X_m, X_m+1) \), where \( X_m+1 \) is given by \((1 - \|X\|^2)^{1/2}\) and ensures that \( X' \) is normalized to a length of 1. (Recall that \( \|X\| \) is the length of the vector \( X \).) Actual inputs to the network must be normalized in this fashion for the network to function properly. (Some software packages either allow or demand that data be normalized to lengths other than 1. Changing normalizations will sometimes help a network overcome learning difficulties.) As a rule, inputs (before normalization) and outputs should be scaled to lie between zero and one.

These networks typically function as lookup tables, in that they memorize input patterns and learn to associate the memorized pattern with its associated output. Like BAM networks, they can completely and exactly learn a data set if there are as many or more Kohonen PEs as there are cases, although they must do the task iteratively. In this case, one Kohonen PE takes responsibility for learning (memorizing) one input vector through competitive unsupervised learning. If there are fewer Kohonen PEs than there are cases, the network is forced to learn some of the statistical properties of the data and will be incapable of exactly learning data. Exact learning will also not occur if one Kohonen PE takes responsibility for more than one input vector. Counterpropagation networks train very rapidly compared to backpropagation networks and generally are good classifiers. They have also been used for continuous property modeling.

Counterpropagation networks have been used in property prediction,\textsuperscript{161} QSAR,\textsuperscript{162} the prediction of retention indices,\textsuperscript{163} and Kovats indices in gas chromatography.\textsuperscript{164} In the latter application these networks performed better than multilinear regression when the coefficient of determination (\( r^2 \)) was low but were worse when \( r^2 \) was high. Apparently, counterpropagation ANNs per-
form better for nonlinear data but lose this advantage as the linearity of the data increases. Two applications to infrared spectra were reported wherein a counterpropagation network learned to produce a digitized spectrum of a compound from a set of structural descriptors. The technique is both accurate and fast enough to be applied to candidate molecules generated from combinatorial chemistry methods. Another study used Kohonen, backpropagation, and counterpropagation networks to determine the geographic and chronological origin of Roman artifacts. These ANNs were found to perform as well as, or better than, clustering methods, PCA, and soft independent modeling by class analogy (SIMCA). Counterpropagation ANNs were used to model the properties of a continuously stirred tank reactor and to predict molecular sequence classifications. A good review of counterpropagation networks and their uses in analytical chemistry may be found in Ref. 170.

**Generalized Regression Networks (GRN)**

Heteroassociative networks that can accept continuous inputs and outputs were invented by Specht. The literature on these generalized regression networks is very limited, even though they appear to be quite powerful. They have been used in the calibration of inductively coupled plasma–atomic emission spectrometry and were found to be comparable to or better than multilinear regression for modeling spectral interferences and matrix effects. GRNs are primarily useful for prediction and system modeling, although they can also be used for classification problems. Their main competitors are backpropagation (including cascade correlation and modular networks) and radial basis function networks. The main advantages of a GRN are that it trains quickly, provides good results even when one is using sparse data, approaches an optimal regression solution as the number of cases increases, and is effective with nonstationary data (data whose statistical properties change with time). They tend to be memory intensive (but no more than a counterpropagation network), although this is usually not a problem unless data sets are very large.

GRNs implement a standard statistical formula for calculating the most likely value for a random scalar variable \( y \) (the output), given a random vector variable \( x \) (the input). This value is the conditional mean of \( y \). (If there is more than one output variable in the data set, the conditional mean of each is calculated.) The conditional mean requires knowledge of the joint probability density function of \( x \) and \( y \). The GRN applies Parzen estimation to estimate these functions directly from the data and in essence encodes these functions in the weights between the two layers of the network. The first layer usually contains as many (or fewer) PEs as there are cases in the data set. This arrangement is analogous to the counterpropagation network and is why very large data sets may become memory intensive. The second layer has one more PE than output variable.
Specht also invented the probabilistic neural network (PNN), which is related to the GRN, but better suited to classification problems. A PNN network has been used in a chromatographic study of aroclor samples. The GRN and PNN warrant further study. Consult Refs. 171–173 for further details of these networks.

**Hopfield Networks**

The autoassociative, single-layer (disregarding input and output layers), laterally connected networks called Hopfield networks can accept either bipolar or binary inputs (depending on the software implementation). We noted previously that a BAM network is a heteroassociative generalization of a Hopfield network, and we would expect many similarities to exist between them. Hopfield networks are good at storing patterns (or, if you like, input data), but their storage capacity is low. Like BAMs, they are trained in one epoch, can exactly store a pattern, and can associate a corrupted input with its uncorrupted counterpart. Unlike a BAM, however, the network weights do not change during the association. Chapter 4 of Ref. 19 gives an explicitly detailed description of Hopfield networks.

Applications of Hopfield networks are limited. One interesting infrared spectral interpretation study used a Hopfield network in a feedback loop to a backpropagation ANN, causing it to train more quickly.

**Kohonen Self-Organizing Map (SOM) Networks**

Unsupervised networks that accept continuous inputs, SOM networks attempt to topologically map from $n$ dimensions ($n$ is the number of input variables) to two dimensions. Each of the two dimensions may be considered to be a feature (i.e., variable) of the data. A topological map is one which preserves order; in this case if two input vectors are close together, then their mapped representations will be close together in the two-dimensional space. SOM networks are useful for visualizing high-dimensional data and are therefore good network choices for clustering.

SOM networks consist of an input layer (one PE for each variable or, if input vectors are normalized, one additional PE) and a Kohonen layer, which contains an arbitrary number of PEs. The Kohonen layer is usually represented as a rectangle or square but is otherwise analogous to the Kohonen layer in a counterpropagation network. In fact, a counterpropagation network, without the Grossberg outstar layer, really is an SOM. PEs in the Kohonen layer compete as in a counterpropagation network, but to make the resulting map smooth, weights of PEs surrounding the winning PE are also modified. The surrounding PEs define a neighborhood, and the size of the neighborhood is an adjustable
parameter. Usually, the neighborhood is taken as the set of nearest neighbors. An SOM will typically train in about 30 epochs. The Kohonen layer almost always contains fewer PEs than there are input cases. This number may need to be modified to get a “good” map. Most implementations also contain parameters that favor PEs that have not recently won, helping to ensure that all Kohonen PEs are used. These parameters vary in their definition; if you are using commercial software, consult the documentation. Chapter 6 of Ref. 17 contains an explicitly detailed discussion of SOMs.

SOMs have been used to identify two-dimensional fluorescence spectra. Linear methods (Laplace and Fourier transforms) and nonlinear methods (moments and modulation functions) work well, except when spectra are very similar or noisy, in which case the SOM is reliable enough to be used as an automatic identification system. SOMs were found to be slightly better at classifying mass spectral data than KNN (K nearest neighbors, a statistical classification method based on the number K of nearest neighbors to an input pattern). Airborne particle source have been successfully classified from their multielement trace patterns and also from their scanning electron microscopy images. In the latter case the SOM gave a more easily interpretable visualization of data than the standard statistical methods of PCA and multidimensional scaling. The extraction of rules for classifying energy dispersion X-ray spectra was made possible using an SOM, as was the online classification of steels. Finally, SOMs have been used successfully in detecting structural similarities between molecules, in identifying aromatic substitution patterns in infrared spectra, in the classification of Roman artifacts based on chemical analysis, and the classification of chemical reactions. Reference 188 is a review of the uses of SOMs in drug design, and Ref. 170 is a review of uses in analytical chemistry.

Perceptron Networks

Perceptron networks are feedforward, heteroassociative (or may be auto-associative) networks that accept continuous inputs. Within the last five years there have been no chemical applications of perceptrons; applications before that time are now largely outmoded by the advent of more powerful ANNs. We mention them briefly for three reasons: they have historical significance, they are ubiquitous in neural network texts, and you will find papers that claim to use perceptrons but in actuality do not.

The history of perceptrons was briefly outlined in the Introduction. They were one of the first ANN paradigms, and although they have significant shortcomings (they really can solve only linear problems), they contributed to the framework of many current ANNs. Many ANN texts begin with a discussion of perceptrons and then proceed to build on that foundation. To that end, a simple perceptron network is shown in Figure 8. Some authors claim to use “per-
Different Types of Artificial Neural Network

The middle layer is either fully or randomly connected to the input layer.

Radial Basis Function (RBF) Networks

RBF networks were invented by Moody and Darken. They are feedforward, heteroassociative (or autoassociative) networks with continuous inputs and outputs. In their basic form they consist of input and output layers (with number of PEs equal to the number of input and output variables), and a hidden layer, which is called either the pattern unit or the prototype layer. The number of PEs in the HL is determined during training. In some ways RBF networks are like backpropagation networks, the main differences being that a Euclidean summation function and Gaussian transfer function are used in the HL of the RBF networks. (The radial basis functions are the Gaussian transfer functions.) The output layer is trained like the output layer of a backpropagation network. The HL is trained in two phases. The first phase typically uses an adaptive K-means algorithm, which is essentially unsupervised competitive Kohonen learning. During this phase the number of HL PEs is determined, along with the centers of the Gaussian functions associated with these PEs. The second training phase determines the widths of the Gaussians. The widths are the root-mean-square distance of the given center to the K-nearest-neighbor centers. For more details, see Refs. 189 and 190.

RBF networks can be used in most situations for which you would also consider a backpropagation network. However, they train more quickly (the HL usually trains in 30–40 epochs) and usually generate better decision boundaries in classification problems. This leads to better generalization, especially when training data is sparse or unrepresentative. On the other hand, HL training is unsupervised, leading to the potential loss of information. Regression problems may require an unbounded transfer function (the Gaussian is a bell-shaped curve).
RBF networks have seen few chemical applications.\textsuperscript{191–195} They were better than multiple linear regression at predicting boiling points from structural parameters.\textsuperscript{191} (Reference 191 also contains a good description of the RBF method.) RBF networks resulted in better calibration than partial least squares in the determination of blood glucose by near-infrared spectroscopy.\textsuperscript{192}

**Recirculation Networks**

Introduced as a plausible alternative to autoassociative backpropagation,\textsuperscript{196} recirculation networks have an input, visible (really the first HL), hidden (really the second HL), and output layer. The input, visible, and output layers all have the same number of PEs, whereas the hidden layer typically has fewer, because data compression is the primary objective of this network. The input layer is correspondingly connected to the visible layer and the hidden layer is correspondingly connected to the output layer. The visible and hidden layers are fully connected in both directions. A pattern moves from the visible layer (time 0), to the hidden layer (time 1), back to the visible layer (time 2), then back to the hidden layer (time 3), and finally out of the network. Unsupervised learning occurs after the second pass (time 2) through the network. The output of the hidden layer at time 1 is the compressed version of the input vector. The output of the visible layer at time 2 is the reconstructed version of the input vector. Learning attempts to reduce the error between the input vector and the reconstructed vector. All summation functions are weighted sums, all output functions are direct. Transfer functions are linear for the input and output layers and sigmoid for the other layers. The input and output layers have no learning rule, while the other two layers use the cumulative delta rule. Recirculation networks contain a learning coefficient and a regression parameter. Their definitions appear to vary depending on the implementation; check the documentation of the respective programs for recommended values. For more details, consult Ref. 196.

Recirculation networks have been used to show that data compression is possible in RNA and DNA sequences.\textsuperscript{197}

**Miscellaneous Networks**

Finally, we briefly mention several other ANNs that have seen limited chemical applications. A connectionist hyperprism ANN has been used in the analysis of ion mobility spectra.\textsuperscript{198} This network shares characteristics of Kohonen and backpropagation networks. The DYSTAL\textsuperscript{199} network has been successfully used to classify orange juice as either adulterated or unadulterated.\textsuperscript{200} A learning vector quantizer (LVQ) network has been used to identify multiple analytes from optical sensor array data.\textsuperscript{201} A wavelet ANN has been applied to the inclusion of β-cyclodextrin with benzene derivatives,\textsuperscript{202} and a
potential function based ANN has been used to classify complex chemical patterns. Genetic algorithms have been combined with backpropagation networks to produce optimal structural descriptors and to optimize network training.

---

**PRACTICAL CONSIDERATIONS IN SOLVING PROBLEMS WITH NEURAL NETWORKS**

By now you should have a good idea whether your analysis problem is a candidate for ANNs. You have concluded that your problem falls into one of the categories outlined earlier (see What Can Neural Networks Be Used For?) and that for one or more reasons also discussed earlier you want to use an ANN, either instead of or in addition to a statistical or artificial intelligence method. You may have some ideas based on the preceding sections about what type of ANN to use.

In this section we show how to successfully develop an ANN application. We present the procedure in serial order, but bear in mind that much of the material in this section, as well as the next two sections, is highly interrelated; you may need to consider two or more steps in the following procedure (or refer to the next two sections) at the same time. Also, remember that we are presenting a general procedure that is applicable to most problems. Specific applications often have suble nuances, and it would be impossible to discuss them all here. Study work in the area of your application to see what has and has not worked. We assume that you already have raw data, that you know how reliable it is, that you know when and/or how to get more data if needed, and that you are thoroughly familiar with your data, the chemistry behind it, and what you are trying to accomplish by analyzing it. If you do not have a good handle on your data, you simply will not be in a position to judge the utility of the application you have developed.

**What Type of Network?**

At this stage in tackling a problem neither you nor anyone else can choose the type of network that will work with 100% certainty. You probably can, however, make a list of several network types that appear to be viable candidates. Unless your objective is to develop a completely new use of a network type, make sure the ANNs on your list have been used to solve problems in the same general category as yours. For example, if yours is a mapping problem, your list should contain ANNs that have successfully solved mapping problems. Make sure the ANNs are auto- or heteroassociative, as required, and that they
are compatible with your data; that is, do not try to force continuous valued data into a network designed for binary inputs. These may seem like obvious points, but experience shows that it is easy for a novice (and “experts”!) to become a bit overwhelmed and muddled by the choices to be made. Resist the temptation to consider only one or two types of network just because you know more about them. Let your problem drive your choice of networks. Do not let a network or your familiarity with a limited number of networks drive your problem. Also, avoid selecting a network type just because it may appear that “everybody” uses that network.

If you have difficulty compiling a list of candidate ANNs, a good general choice would be a standard backpropagation network, since these have been used with at least fair success for a wide variety of problems. If nothing else, it will serve to get you acclimated to your analysis problem. However, one of the strengths of the backpropagation network—its general applicability—may also be a liability. Some view it as a “jack of all trades but master of none” ANN. We would be delighted if the ANNs discussed in the preceding section could, or would, solve all chemical data analysis problems. They will not; so remember that other ANNs exist. Finally, prepare yourself for the possibility that your carefully and thoroughly selected network may not work. It is not uncommon to try a few ANNs before finding one that yields satisfactory, or superior, results.

**Data Preprocessing**

Rarely will your raw data be in an acceptable form for input to an ANN. You must perform some combination of transforming and scaling to both the input and output data. Preprocessing is often the single most important operation in the development of a successful application, and you are enthusiastically urged to pay close attention to it. Time spent here will more than repay itself later. Remember: garbage in, garbage out.

Before you can make any intelligent decisions about preprocessing, you must know your data. You should do some exploratory data analysis on all variables (input and output). What is the mean of each variable? Are the values of each variable evenly distributed over their range, or do they tend to cluster around one or more values? Do any variables have apparent outliers (values that seem to be well outside some normal range)? Are any variables interrelated, either functionally or with strong positive or negative correlations? You probably do not need to compile vast numbers of descriptive statistics, but you do need a good qualitative feel for the data. A good way to do this is to visually inspect plots of the data. Almost any modern statistics or visualization software package can be used to do this; several are given in Refs. 209–212. (A good source of information for a wide variety of commercial software is *Scientific Computing and Automation* magazine.213) This is also a good time to ask if variables can be grouped in some way. That is, do you have several variables
that are measures of similar or related quantities or are expressed in the same units? If so, do the values of these variables all fall in the same range or not? If you have a supervised classification problem, think about how you define categories. Try to have as few categories as possible. The more categories you have, the more likely it is that the membership of each will be smaller and that you will need to obtain more data. (In the olive oil example, you may have run into trouble if, instead of categorizing oils by geographic region, you categorized them by county or province, each of which may have lacked sufficient cases to enable an ANN to learn effectively.)

The issues raised by the questions above can be addressed by three types of operation: transformation, scaling, and encoding. We present brief summaries of each. For more details, refer to any of several excellent texts.214–217

The idea behind transforming data is to force the data into an even distribution; the values of a variable should be relatively evenly spaced over the range of the variable. Outliers may swamp out important variations of other cases. Data that tends to cluster around two or more values may mask important information within each cluster.

A few general rules should cover most situations you will encounter. Outliers may be removed with a compressing transformation such as the log (an extreme compressor) or square-root (a mild compressor). If data values are sometimes negative and this sign difference is important, a cube root may be a suitable transformation. If the sign is unimportant, you may simply be able to add a constant to all data values and then perform a log or square-root transformation. If a variable exhibits two or more clusters, histogram equalization may be required.216,217 This method is also useful for transforming data with discontinuities, which are, in effect, clusters. It is beyond the scope of this tutorial to discuss in detail histogram equalizing, but it is perhaps the most foolproof transform available. Histogram equalization is computationally intensive and may suffer from information loss, since data is placed in "bins." The latter disadvantage may be overcome by using enough bins (at the expense of increased computation time).

A note about outliers is appropriate here. Many data analysis methods, including least-squares regression and backpropagation ANNs, are sensitive to outliers; that is, the methods are not robust. This is because they rely on minimizing a function of squared errors, so the outliers are too influential. Some recent work attempts to make backpropagation methods robust.218–221 (There are also robust statistical methods.) Nonetheless, we strongly encourage you to study your data and make appropriate transformations.

Most linear methods are immune to scaling. Most nonlinear methods, including virtually all ANNs, are not, because they assume that variables having large variation are more important than those having small variation. This is true for both input and output variables. For example, an ANN that trains by minimizing an error over all output PEs may spend an inordinate amount of time reducing the error of output PEs associated with variables of high variance,
simply because this is the easiest place to reduce error. Unfortunately, output 
PEs with small variance may be largely ignored (see later section on Perfor-
mance Metrics). Although we have stated that a strength of ANNs is their 
ability to model several output variables at once, you may want to think se-
riously before trying this unless you are sure the ANN you have in mind will 
treat all output PEs fairly. To ensure this, all output variables should be scaled to 
have the same variance. Of course, if you scale them to have equal variance, you 
may have destroyed useful discriminating information! Our advice is to model 
only one output at a time unless you are willing to wade through a variety of 
performance metrics in trying to determine why some outputs give great results 
while others do not.

Input variables with widely different scalings may cause training problems 
if they require that some network weights be large in magnitude; many ANNs 
have difficulty adapting their weights over a wide range. If an input variable has 
a large average value, any bias terms a network has may need to be quite large, 
which again is difficult for some ANNs to achieve.

Ideally, variables should be scaled so that variation reflects their relative 
importance. Relative importance is often quite difficult to determine (presum-
ably, if you knew this, you would be a long way toward solving your analysis 
problem), and in the absence of other information it is usually best to scale all 
variables in the same way. There are a variety of ways to do this, but remember 
to scale only after any required transformations have been done. One way to 
scale is to calculate a Z-score or standard score.\footnote{[62,216,217]} For a given variable $X$, 
the Z-score for the $i$th case is

$$Z_i = (X_i - X_{av})/S$$

where $X_{av}$ is the average of $X$, and $S$ is the standard deviation of $X$. The Z-scores 
have a mean of zero and a standard deviation of one. Another possibility is to 
linearly scale all values for a variable to lie between two numbers. For example, 
to scale between 0 and 1, simply calculate $(X_i - X_{\text{min}})/(X_{\text{max}} - X_{\text{min}})$, where 
$X_{\text{max}}$ and $X_{\text{min}}$ are the maximum and minimum values of $X$.

The two scalings above are general suggestions. Some networks require a 
certain scaling. For example, a probabilistic neural network requires that input 
variables have a mean of zero and a maximum length of one. Some people will 
claim that Z-scores are not appropriate input for backpropagation networks 
and that only scaling between zero and one (or 0.1 and 0.9, etc.) is appropriate. 
The reasoning behind this is that a sigmoid transfer function squashes effective 
inputs so they will lie between zero and one. A hyperbolic tangent transfer 
function squashes effective inputs to lie between $-1$ and $+1$, and the same 
people who insist on scaling between, say, zero and one would argue for scaling 
between these two values for a backpropagation network with a tanh transfer 
function. There seems to be some general disagreement on this point, but fur-
ther discussion is available elsewhere (see Ref. 62, pp. 70–74). The literature on backpropagation applications suggests that scaling between zero and one (or −1 and +1) produces satisfactory results.

One other issue regarding scaling should be mentioned. Do you scale each variable separately, all variables together, or some separately and others collectively? If all members of a group of variables are measuring similar quantities (in the same physical units) you may wish to scale this group collectively. Variables which are measuring quantities of different types usually should be scaled individually. However, these are not fail-safe guidelines. Again, refer to pp. 70–74 of Ref. 62 for further discussion.

You should not be surprised if you need to fiddle with your data. Scaling has few hard and fast rules, and there is some art to it. Unless your ANN demands a particular scaling, we suggest that you start with something simple—say scaling between zero and one—continue with the development procedure, and assess the training and testing results. If they are not satisfactory, try to determine why (we will talk about what to do when things go wrong later in this section). It could well be that poor results are not related to scaling. If they are, try something more sophisticated.

All the preceding discussion has concerned continuous numerical data. Not all data is of this form, and we now discuss the encoding of nonnumerical data. Logical variables have one of two values that are encoded as zero and one. An example is the presence or absence (a yes or no) of a feature. Typically the presence (yes) is encoded as a 1 and the absence (no) as a 0. Categorical data should be encoded in some variation of a one-of-N code (N equals the number of categories) or a thermometer code.

Suppose we have four categories A, B, C, and D, which are mutually exclusive. In a one-of-4 code, cases are encoded with four variables, the first denoting the presence or absence of membership in category A, the second denoting the presence or absence of membership in category B, and so on. For example, a case belonging to C would have the values 0, 0, 1, 0, respectively, for each of the four variables. Fuzzy one-of-N codes are used to preserve some sort of relationship between neighboring categories. A nonchemical example is the following. Suppose the days of the week are categories and that Sunday is category one, Monday is category two, and so on. A fuzzy one-of-7 code for Wednesday is 0, 0, 1, 1, 0, 0. Notice that in addition to the fourth variable being 1, the variables on either side are also 1, indicating that Wednesday is next to Tuesday and Thursday. In a gradient one-of-7 code, Wednesday might be denoted as 0, 0.25, 0.5, 1, 0.5, 0.25, 0. Why not use just one variable for seven categories, allowing it to take the values 0.142 (category 1), 0.284 (category 2), etc.? Experience has shown that usually this simply does not work. The only exception is a two-category problem, which may be encoded in one variable whose value is 0 for category A and 1 for category B. (To do this, A and B must be mutually exclusive.)
Thermometer codes allow you to establish a ranking among categories. Suppose there are three models of some product: the base model, the standard model, and the deluxe model. These would be encoded with three variables, the base model being given by 1, 0, 0, the standard model by 1, 1, 0, and the deluxe model by 1, 1, 1.

Variable Selection, Reduction, and Orthogonalization

On occasion you will potentially have more variables than are needed to solve a problem, although you may not know it at the time. Generally, you would like to use the lowest number of variables consistent with satisfactory network training and testing. This reduces development time because training times are shorter, the total number of PEs lower, and any analysis of the network easier. There are several ways of eliminating variables. Pruning methods are one way to eliminate variables, and these are discussed later in the section on Analysis of Neural Networks. Here we mention two ways of computing new variables that are linear combinations of old variables. The advantage of doing this transformation is that usually fewer new variables are required for a given level of performance. The disadvantage is that in going from the old set of variables to the new, some information is lost, and it may be just the information you need.

We have already mentioned principal components. For our purposes it is enough to say that these new variables are linear combinations of the old variables. The new variables are orthogonal to each other; that is, they are completely independent of one another in that there is no redundancy in the information they carry. They are uncorrelated.

For classification problems, a better set of new variables (compared to principal components) consists of linear discriminant functions. The maximum possible number of linear discriminant functions is one less than the number of categories, and the use of such functions often makes possible a dramatic reduction in the number of variables in certain problems. The other advantage is that under certain conditions (the distribution within each category is similar for all categories) the functions optimally separate categories, at least in a linear sense. Their use is highly recommended in classification problems, but should never be relied on solely. Since the functions are linear, information that inherently nonlinear ANNs could use to great effect has been discarded, and you are advised, particularly in the beginning phases of training, to use them in addition to other carefully chosen variables. For more details on either method, see Ref. 223.

More recently, a new statistical method—correlative component analysis (CCA)—has been introduced. This method is somewhat analogous to principal components analysis, but the correlative components are constructed to enhance subsequent classifications by containing more information than do principal components.
Principal components have been used as input to backpropagation networks with mixed results.\textsuperscript{100,116,121,225,226} In an application to ion mobility spectrometry, training time was reduced.\textsuperscript{121} In an application to calibration of near-infrared spectra, the standard error of prediction was reduced by 50–75\% compared to that obtained from principal components regression and partial least-square techniques.\textsuperscript{100} When applied to nonlinear modeling of chemical data, principal component inputs resulted in improved training times and accuracy of prediction compared to not using principal components, although for noisy data there was no improvement.\textsuperscript{225} Use of principal components resulted in a decrease in performance from 96\% to 90\% correct on a test set when jet fuels were classified from laser-induced fluorescence spectra.\textsuperscript{116}

**Training and Testing Sets**

Now that you have an idea of which network(s) you will use and have preprocessed your data, you need to construct a training set and a test set. The training set should contain cases that are representative of the entire population of cases. Therefore, the training set cases should be varied enough to permit each of the input variables to take values over nearly its entire range. For modeling, mapping, and association problems, this requirement will likely result in outputs that also vary over their ranges. For unsupervised clustering problems, this situation would hopefully result in enough variation to reveal the underlying cluster structure of the data. (It is a good idea to look at the clusters you obtain from an ANN and see if you can make sense of them based on chemical and physical considerations; this will help you determine whether the underlying structure has been found. Alternatively, you can cluster data for which you know the correct categories to see how the ANN is performing.) For supervised classification problems, however, it is usually best to pay more attention to the categories (outputs). For training purposes, it is usually best to have equal numbers of cases in each category regardless of whether this matches the actual distribution of data. If there are too few cases in a category, the network will not have enough opportunities to learn to make reliable predictions. Kohonen\textsuperscript{40} and Rumelhart and McClelland\textsuperscript{49} discuss in more detail the need for using a representative training set.

The guidelines above should be followed if possible, but you should be aware that in real-world applications it may be difficult or impossible to meet them. For example, certain categories just do not occur very often. In such cases you will need to use some of the performance metrics discussed later in this chapter for assessing reliability of the ANN. Familiarity with your data is also important here. Do the values of your variables really range over their expected values for the cases in your data set? If not, you may want to consider gathering more data.

The test set should contain only cases that are not in the training set. If a test set case is also in the training set, you will merely be testing the ability of the
ANN to memorize that case. The test set should also contain a representative sampling of cases to realistically assess how the ANN responds to new situations. A word about autoassociation problems is in order here. If your goal is to use an ANN to simply store patterns or to compress data, you really do not need a test set because all you care about are the cases with which you train the network. If you want to pass corrupt data through the ANN to see if the network will output a clean version of the input, you may want to construct a test set to see how well the network can do this; training set construction is presumably trivial here: you know what data you want to store or compress, and this data is the training set.

In practice, how do you actually go about creating a training set that meets the guidelines above as closely as possible? If your data set is small enough you may be able to handpick the cases to place in the training set. With small data sets, it may be difficult to form a representative training set and still have enough cases to form a realistic test set. If your data set is too large for this to be feasible, thoroughly randomize the cases and then randomly select cases to make up the training set. The question here is, What percentage of cases should go in the training set? You may need to answer this by resorting to trial and error. To see if you have formed a representative training set, pick a percentage, form the training set, and perform some exploratory data analysis (calculate the mean, range, and standard deviation; then make plots of the variables for the training set, and compare these to the same quantities for variables in the complete data set). As with the handpicked scenario, you may not have enough cases for a good test set. If not, you may wish to form several training and test set pairs, and then train and test a network on each pair, averaging the results. A Kohonen self-organizing map has been used to generate a training set by placing one case from each generated cluster into the training set, with all other cases placed in the test set.227 This method of forming training sets is becoming increasingly popular. Another method involves the use of simulated data to form a training set.220 The method is fairly complex (although easily programmed on a computer) and appears to be somewhat application specific (e.g., time series data). However, similar methods might be useful in other applications. A good review of methods for forming training and testing sets is in Ref. 228.

Bayesian regularized backpropagation ANNs229 have recently been introduced to QSAR problems.230 It has been suggested that no test set is required for these networks.

**Training the Network**

You are now ready to set up and train the network. To set up the network, you need to select the number of layers of PEs, the number of PEs in each layer, and the values for any adjustable parameters (learning rates, etc.). For many ANNs the architecture question is almost trivial; there is a well-defined number of layers and a well-defined number of PEs in each layer. In all cases, however,
the number of input PEs will be equal to the number of input variables, or one more than that if an extra term is required to normalize the input patterns to a given length. For supervised classification problems, the number of output PEs is determined by the category encoding, while for unsupervised clustering problems the number is determined primarily by the number of desired clusters. A trial-and-error approach may be required here. For other types of problem, the number of output PEs is equal to the number of input PEs (autoassociative), the number of variables you are trying to predict (modeling), or the number of dimensions to which you are reducing or compressing data (certain mapping problems).

Determining the number of hidden layer PEs to use is not as easy. Although some networks specify a certain number of HL PEs, be aware that often it is possible to change this number to force a particular behavior on the network. An example of this is a counterpropagation network that "requires" the number of HL PEs be equal to the number of training set cases. If set up in this way, the network functions as a look-up table, with each HL PE memorizing one input case. However, if there are fewer HL PEs than training cases, some HL PEs will learn the average of two or more training cases. This, in effect, causes the network to learn statistical properties of the training set.\textsuperscript{162} For networks in which the number of HL PEs is not specified, a trial-and-error procedure must often be used. Train and test several networks, each having a different number of HL PEs; the network with the best test results usually is the network of choice. (Of course, you want to avoid overtraining when doing this; see discussion later in this section.)

There has been much discussion regarding the number of HL PEs in backpropagation networks. If there are too many total weights in the network, the network will simply memorize cases and therefore perhaps perform poorly on a test set. This undesirable outcome is referred to as overfitting and is analogous to fitting, for example, 10 data points with a ninth-degree polynomial. This polynomial has 10 constants which are uniquely determined by the 10 points. One general rule is that the number of HL PEs should be such that there are three times as many training cases as total weights in the network. This issue is briefly discussed in Ref. 22. You should note that the concept of overfitting is not applicable to all ANNs. Returning to the counterpropagation network, we observe that this ANN, when configured with the same number of HL PEs as input cases, is designed to exactly memorize each input case.

Most ANNs have a required number of layers, each performing a specific role. The backpropagation network allows some flexibility in the number of hidden layers. Generally you should use one HL, although it is possible that two Hls will perform better if the total number of network PEs is less than for the one HL cases. Often, the best way to decide how many Hls to use is simply to experiment.

Most ANNs have specific transfer functions that must be used in a given layer. Once again, the backpropagation network is an exception. Whereas the
sigmoid and hyperbolic tangent functions are most common, any differentiable function may be used. The choice between sigmoid and hyperbolic tangent functions normally would be made by trial and error. (Some workers suggest preprocessing inputs between $-1$ and $+1$ for tanh, and 0 and 1 for sigmoid.) An interesting discussion of transfer functions is given by Wang et al.\textsuperscript{231} and Muñoz-Caro and Nino,\textsuperscript{149} who try to choose a transfer function that describes nonlinearity specific to a particular problem. Schulze et al.\textsuperscript{118} found that sine functions in the HL resulted in faster training and better discrimination between closely related Raman spectra, but use of these functions was less tolerant to spectral distortions. Harrington\textsuperscript{232} offers a discussion of sigmoid and hyperbolic tangent functions, whereas Li et al.\textsuperscript{233} recommend several transfer functions for classification, nonlinear fitting, and linear fitting problems.

Finally, all ANNs have parameters that must be set by the user. These include learning rates, gain and offset in some transfer functions, and ranges between which initial values of weights should fall. Where possible, suggested values for these parameters for specific networks were given in the preceding sections. These were merely suggestions, and values may need to differ from these to optimize performance. Parameter selection may be done by trial and error, although a potentially better way was suggested by Munk et al.,\textsuperscript{96} who used the simplex method to optimize parameters in a backpropagation network in an application to multispectral infrared interpretation.

At this stage, you are ready to train the network. If you are using a backpropagation network, you should randomize the order of cases in the training set and change the order after every few epochs. This will help counter the tendency of these networks to learn the first cases better than later ones. We suggest that you use this randomization scheme no matter what network you use.

When should you stop training? The answer is application and network specific. Some ANNs require only one epoch, after which they are completely trained (a Hopfield network is an example), and the question is moot. For networks that learn iteratively (more than one epoch required), training should be stopped after the weights settle down (i.e., converge) for unsupervised learning. (You should perturb the weights at this point, and then continue training a while longer and see if the results change; if not, you are probably OK.) For autoassociative networks in this category, training should be stopped when the network has memorized the input patterns to a tolerance sufficient for your purposes (i.e., inputs match outputs to a prescribed tolerance.) If you want such a network to reconstruct corrupt patterns, however, training should be stopped when the test set of corrupt patterns (incomplete or noisy patterns) has been reconstructed to a prescribed tolerance. Alternatively, you could use a pre-specified value of some performance metric. For all other cases, you want to train enough to learn the essential characteristics of the data but not so long that performance on the test set drops. Typically you will find that as training progresses, performance on the training set will increase and then level off, whereas test set performance will increase and then decrease. When test set
performance decreases, the network is becoming overtrained. You want to terminate training when test performance is at a maximum. Another strategy that is sometimes used is to form three sets: a training set, a test set, and a validation set. The validation set is treated in the same manner as the test set in the paragraph above. Training is stopped when the performance on the validation set begins to worsen. The trained network is then used on the test set. Typically you would then use some performance metric or metrics (see later section on Performance Metrics) to quantify performance for each of the three data sets.

**Learning Versus Generalization**

The issue at the end of the last paragraph is the ability of an ANN to learn versus its ability to generalize, that is, to perform successfully when presented with new inputs. Tetko et al.\(^{234}\) compared overfitting and overtraining for one HL backpropagation network and concluded that overfitting has no influence on prediction ability if overtraining is avoided by cross-validation.\(^{235,236}\) The latter is a process whereby all but one case in a data set are used to train a network, which is then tested on the excluded case. This process is repeated until all cases have been individually tested. These calculations are also known as jackknife or leave-one-out calculations. Performances on the one-member test sets are averaged, and training is terminated when the average test set performance reaches a maximum. A common variation on leave-one-out calculations is leave-\(n\)-out calculations; \(n\) is typically some relatively small integer. This is a useful approach if the computational overhead of leave-one-out calculations is too great, as it sometimes is for larger data sets.

We have seen that there are three strategies for avoiding overtraining: the validation set, the leave-one-out, and leave-\(n\)-out methods. There appears to be no definite consensus on which method is best, although as data sets become smaller, the leave-one-out method is usually preferable, whereas as data sets become larger, the validation set method is usually preferable. For more details on these methods, consult Refs. 22, 29, 216, 217, 223, and 228.

For a more thorough discussion of generalization, see Chapter 6 of the book by Hertz, Krogh, and Palmer.\(^{13}\) The chapter gives a theoretical discussion of generalization and presents several algorithms for constructing feedforward networks with optimal (in the sense of giving the best generalization) architectures. Although some of these algorithms appear to be unknown to the chemical community, they would be quite useful in, for example, eliminating a trial-and-error approach to finding the best number of HL PEs.

By now you may be thinking, “Can an answer be obtained easily and quickly?” There are many things to consider in setting up an ANN application, but in our experience many problems can be solved quite effectively with simple scaling of the data, a reasonable guess for the number of HL PEs, randomly chosen training and test sets, and a “generic” set of parameters. Refinements on this simple approach become necessary if you are trying to squeeze the last bit of
performance from the ANN. From time to time, however, an ANN will not perform the way you want it to. Rather than start the whole application procedure over again, there are a few things to look at that often will get the ANN back on track. We focus primarily on backpropagation networks, but also indicate what steps might be taken with other ANNs.

A backpropagation network tries to minimize the total squared error of the output PEs. It is recommended that you monitor this quantity as training progresses (regardless of which type of ANN you are using and even if you are determining when to stop training by monitoring a different performance metric applied to a test set). Sometimes you will find that this quantity decreases very slowly with training, an indication that the network is not converging.

First, check to see that your training cases are truly randomized for all variables. For example, even if the training set has been randomized for the output variables, it is possible that there are still "clumps" of cases, where one or more input variables have very similar values. All delta learning rules have a tendency to learn rules at the beginning of a data set and forget them as they learn rules at the end of the set. A thorough shuffle of the training cases after every few epochs is strongly recommended for ANNs of all types.

Second, verify the effectiveness of any transformations and scalings you performed on the raw data; these should be few, if any, outliers, and so on. (Of course, you should have checked this before, but this is a good time to recheck.)

Third, check for any abnormally large values (positive or negative) of the network weights. This is an indication that PEs are "saturated," especially if such anomalies appear relatively early in training. (Saturated PEs are usually OK late in the training process if the network otherwise appears to be converging properly; having saturated PEs late in the training process is an indication that some rule has been learned.) Once saturated, however, PEs tend to learn slowly if at all. Try reducing the values of the learning coefficients. Large coefficients lend themselves to potentially faster learning but at the risk of saturation, nonconvergence, and settling into a local instead of global minimum in the squared error. Small coefficients tend to overcome these risks at the price of slow learning. If training time is important, the trick is to find a satisfactory middle ground for the coefficient values. Suggested values for learning coefficients given earlier in the section on Processing Elements will not work well for every data set. Also, as a general rule, the larger the data set, the smaller the coefficients should be. It is a good idea to check for saturation and do some tinkering with learning coefficients no matter what type of ANN you have. After you make any of the adjustments suggested, start training from scratch; if, for example, you have saturated PEs, it may not be possible to unsaturate them simply by continuing to train with reduced learning coefficients.

Fourth, check the initial weight values of the network. Typically, before any training, the weights may take random values between -0.1 and +0.1 (this may vary, depending on network type). If the initial weights are too large, the network may quickly move to a saturation state. Typically, the more data you have, or the larger the network, the smaller the initial weights should be. Here
again, there is a bit of a trade-off: if the weights are too small, training time may increase.

If the squared error drops during the initial phases of training but then levels off at what you think is too high a value, check the foregoing points after you have perturbed the network weights. In a backpropagation network it is possible that the weights have settled into a local minimum. Perturbing all the weights by a random amount may be sufficient to "bump" the network out of the local minimum, allowing it to reach either a deeper local minimum (which may be sufficient for your purposes) or, if you are lucky, a global minimum.

Sometimes after having trained two identical networks on the same data, you will find that you have obtained different results. This is usually a consequence of the initial weights being randomized and the ANN having settled into different minima on the weights surface. Two apparently identical networks may have different initial weights. (Typically, whatever software you are using will have a provision for generating random numbers, and you may not have much control over this operation.) Different initial weights may cause each network to have different final weights, even after the same number of epochs. (This is even more likely to happen if input data is shuffled.) You should always check to see whether the final weights are different. If they are, you should use an ensemble of networks for every training and testing run you make and average the results over the ensemble. The number of networks to use in the ensemble is application and data set dependent, but typically the larger the variation in results from network to network, the more networks you will want to use.

Finally, a word about one-of-N codes. If you are monitoring total squared error or root-mean-square error during training, you may be surprised to find that despite a small value for these errors, classification rates are poor for certain categories. This may happen simply because many ANNs find it easy to train an output PE to zero. This is especially true for backpropagation networks. Since most of the outputs for one-of-N codes are zero, it is easy for the network to reduce the total squared error by training all output PEs to zero! The problem is exacerbated when \( N \) is large. If \( N \) is not too large, set the learning coefficients to very small values. Otherwise, consider using a softmax output function (Eq. [23]) in the output layer of a backpropagation network. See Ref. 273 for details.

A review of some of the topics we have discussed in this section as they relate to BP networks is given in Ref. 238. A discussion of training and testing for noisy data with feedforward networks is given in Refs. 239 and 240.

**PERFORMANCE METRICS**

A neural network application is of little use unless quantitative statements concerning its performance can be made. There are a large number of performance metrics (PMs), some of which are application specific, and we cannot be
comprehensive in our coverage of them. Rather, we discuss several PMs that are applicable to different situations and, since most PMs have limitations and weaknesses, we try to elucidate these to demonstrate what you should consider.

Classification Problems

We begin with supervised classification problems and assume that we have training and test sets that were chosen according to the guidelines given in the preceding section. We also assume that the correct categories are known (almost always the case in classification problems in chemistry) and that the output PEs have or can take only the values zero and one. We will discuss later the case of output PEs that can take a continuous range of values, say from zero to one.

The simplest way of evaluating performance is to calculate the percent of correct classifications. This may be done globally (i.e., total number of correct classifications for all categories divided by total number of cases; this is sometimes called the global response quality) or for each category (i.e., total number of correct classifications for a category divided by the total number of cases in the category; this is sometimes called the response quality for a category). However, the global calculation may mask poor performance for one or more categories, particularly if there are different numbers of cases for each category. Suppose we have two categories A and B, each with an equal number of cases; category A is 90% correctly predicted and B is 70% correctly predicted. The global performance is (90)(0.5) + (70)(0.5) = 80% correct. But suppose A had 85% of the total cases. Then the global performance is (90)(0.85) + (70)(0.15) = 87% correct. This is better overall performance but does tend to mask the relatively poor performance for B. Alternatively, suppose A is 85% correctly predicted and has 80% of the total cases, and that B is 65% correctly predicted. The global performance is (85)(0.8) + (65)(0.2) = 81% correct, which is better than the first set, even though performance for both A and B is less. Ideally, training and test sets should have the same number of cases in each category, but this is sometimes difficult to achieve.

Another problem (which is related to the above) with the percent correct PM is its failure to account for the existence of a statistical chance (50% for the two category case) that each case will be correctly classified even if the ANN has not learned at all. If A and B have equal numbers of cases, the global performance is (50)(0.5) + (50)(0.5) = 50% correct based on random chance. (This assumes no learning and a coin flip for each case.) If A has 90% of the cases, the global performance is still (50)(0.9) + (50)(0.1) = 50% correct based on random chance (no learning and a coin flip). However, in this case even an untrained network could achieve a global performance of (100)(0.9) + (0)(0.1) = 90% correct simply by classifying every case as belonging to category A; the a priori probability of membership in A is 90%. Of course, the opposite could happen as well, resulting in 10% correct if the network classified every case as category
B. If you use percent correct as a PM, you should report the percent correct for each category and specify how many cases each category has.

To overcome the shortcomings above, it is useful to introduce the following possible decisions which an ANN can make for one given category A. The ANN can make a true positive (TP) decision. This means the ANN correctly classifies a case as belonging to A. A true negative (TN) decision means the ANN correctly classifies a case as not belonging to A. A false positive (FP) decision is one in which the ANN incorrectly classifies a case as belonging to A (i.e., the case does not belong to A, but the ANN predicts that it does belong to A). Finally, a false negative (FN) decision is one in which the ANN incorrectly classifies a case as not belonging to A (i.e., the case belongs to A, but the ANN predicts that it does not belong to A). TP and TN decisions are correct decisions whereas FP and FN decisions are incorrect decisions. For a single category A, a perfect classifier gives

\[
TP + TN = \text{total cases in A} \tag{41}
\]

\[
FP + FN = 0
\]

We can use the definitions above to introduce PMs that are often more enlightening in describing ANN performance.

Sensitivity = \(\frac{TP}{TP + FN}\). This is also known as recall and as the true positive ratio. This is the number of cases correctly classified as belonging to A divided by the total number of cases that actually belong to A. Alternatively, it is the probability that a case will be correctly classified as belonging to A. It gives an indication of the relative number of false negatives and is an important PM when it is crucial that a case be correctly classified as belonging to A. For example, if a patient has cancer, it is important that the patient be classified as having cancer.

Specificity = \(\frac{TN}{TN + FP}\). This is also called the true negative ratio and is the opposite of the sensitivity. It is the probability that a case is correctly classified as not belonging to A. It is an important PM when it is crucial that this classification be the case.

Positive predictive value = \(\frac{TP}{TP + FP}\). This is also called the precision. This is the number of cases correctly classified as belonging to A divided by the total number of cases classified (both correctly and incorrectly) as belonging to A. Alternatively, it is the probability that a case classified as belonging to A really does belong to A. It gives an indication of the relative number of false positives and is an important PM when an FP is undesirable. For example, if a patient does not have cancer, it is important that the patient not be classified as having cancer.

False alarm rate = \(\frac{FP}{FP + TN}\). This is also called the false positive ratio. Note that it is equal to \(1 - \text{specificity}\). It is the probability that a case is incorrectly classified as belonging to A.
Distinctions between these PMs are sometimes subtle but a little thought will show how they are applicable to your data. All these PMs may range from zero to one, inclusive. The better the classifier, the larger the sensitivity, specificity, and positive predictive value, and the smaller the false alarm rate. These PMs do not appear to have been used much in chemical applications, but have been used in an atomic physics application.\textsuperscript{241} For further details on them consult Chapter 7 of Ref. 62, and also Refs. 242 and 243.

Two other PMs are the \textit{reliability index} \((A_{50})\) and the \textit{Matthews coefficient}.\textsuperscript{244} The reliability index is a measure of the number of false positives. It is defined as follows:

\[
A_{50} = 0.5 \frac{TP}{(0.5 \cdot TP + FP)}
\]  

[42]

Notice that an \(A_{50}\) value of 1 means that the category A cases are correctly classified and that there are no false positives. False positives are counted twice as heavily as true positives. This measure has been used in infrared spectrum interpretation.\textsuperscript{52}

The Matthews coefficient \(C\) attempts to eliminate bias caused by an uneven distribution of cases among categories (i.e., for the two-category case, other than 50% of total cases belonging to each category). It is defined as follows:

\[
C = \frac{(TP)(TN) - (FN)(FP)}{[(TP + FN)(TP + FP)(TN + FN)(TN + FP)]^{1/2}}
\]  

[43]

and ranges between \(-1\) and \(+1\), indicating completely incorrect performance and completely correct performance, respectively, for a category A. If \(C = 0\), then the ANN has learned the percentage of cases belonging to A, but has learned nothing else. This coefficient has been used in secondary protein structure prediction\textsuperscript{110} and in nuclear stability prediction.\textsuperscript{245,246}

Finally we mention two other PMs. The \textit{extrastatistical quality}\textsuperscript{247} is a measure of the improvement of classification quality over that due to chance and has been used in predicting NMR shifts\textsuperscript{114} and in IR spectra interpretation.\textsuperscript{93} The \textit{chi-squared} test is a statistical test that determines whether the differences between an actual distribution and an expected distribution are due to chance. If you know how the cases in a training or test set should be distributed among categories (the expected distribution) and how the ANN distributes the cases (the actual distribution), the chi-squared test is a useful way of comparing these two distributions. See Chapter 7 of Ref. 62 for an example of how this is done.

We next consider what to do if the output PEs can take a continuous range of values. We assume that these values are between zero and one, but what we say could easily be adapted to other values. The output PEs of a backpropagation (for example) network used for classification will likely not have values of
0.0 or 1.0 for all cases (nor will any ANN capable of continuous outputs). Suppose the output PE corresponding to category A gives an output of 0.65 for a particular case. Should we interpret this value as indicating membership in category A? There are several ways of dealing with this question. The simplest way is to somewhat arbitrarily set a threshold value of 0.5 for this output PE and interpret values greater than 0.5 as indicating category membership and values less than 0.5 as indicating nonmembership. Often this will work and give satisfactory results. You may not, however, get the best results—as measured by any PM—in this way. Also, what do you do if categories are mutually exclusive, but two or more output PEs have values greater than the threshold?

In regard to obtaining the best results, we make the following observations. Often you will find that as the threshold approaches one, the sensitivity, positive predictive value, false alarm rate, and proportion of all cases correct will decrease while the specificity will increase. By raising the threshold, you are making it more difficult for the network to learn and produce true positives. The positives it does learn are more likely to be true positives. You should try several thresholds and plot the PMs of importance to you versus the threshold. Of course, you should do this for each category, and you may also want to average the PMs over a series of jackknife calculations. This procedure should give you a good idea of what threshold to use.

The problem of two or more output PEs having values greater than threshold may be partially alleviated by the foregoing process. Or you may decide that the PE with the largest output is the winner and classify the case into the corresponding category. Another alternative was proposed by Ricard, Cachet, and Cabrol-Bass. In an application to infrared spectra interpretation, these authors determined thresholds by minimizing a cost function that depended on the number of correct minus the number of false classifications as determined on a test set. This resulted in two thresholds. Outputs greater than the large threshold corresponded to category membership, and those less than the small threshold corresponded to category nonmembership; cases giving outputs between the two thresholds were deemed unclassifiable. In general, the values for the two thresholds were different for different categories.

Once you have decided what threshold(s) to use for a category, all the PMs above are applicable. Every output greater than threshold is assigned the value 1.0, and every output less than threshold is assigned the value 0.0.

There are PMs that do not require assigning outputs to zero or one. These are most often used in deciding when to stop training. The idea is to monitor these PMs as a function of epoch for the training and test sets. The epoch for which the PM is a minimum or maximum (depending on the PM) for the test set (or validation set, or average over jackknife test sets) is the epoch at which training is terminated. A question that arises here is whether you should use a global PM or a PM for each category. A global PM may reach its maximum or minimum at a number of epochs different from that required for a single category PM. And each of the single-category PMs may reach its maximum or
minimum values at different numbers of epochs. If the maxima or minima occur at different epochs, then stopping training at a minimum for one PM obviously will not necessarily be a minimum for another PM. What can then happen is that the ANN may generalize quite well for one category but be overtrained (and therefore not generalize well) for another. So, when do you stop training? Ultimately, it is your decision to make, and after first checking to see whether there are generalization problems for some categories, you will need to consider whether in fact you need equal generalization for all categories, whether all categories are equally important, and so on. Of course, any of the earlier PMs could be used in the foregoing ways as well.

The apparent motivation for using these alternative PMs is that they are functions of output errors and are closely related to error functions that are minimized by some ANNs during their learning phase. In our opinion, you should use PMs designed for outputs of zero and one when working on classification problems. We discuss only one PM designed for classification problems using ANNs with continuous valued output PEs. The discrimination index\(^{114}\) for one category \(A\) is defined as follows:

\[
DI = \frac{(M - N)}{\sqrt{(PV + QW)/R}}^{1/2}
\]

where \(M\) is the mean of the outputs for cases belonging to \(A\), \(N\) is the mean of the outputs for cases not belonging to \(A\), \(P\) is the number of cases belonging to \(A\), \(V\) is the variance of the outputs for cases belonging to \(A\), \(Q\) is the number of cases not belonging to \(A\), \(W\) is the variance of the outputs for cases not belonging to \(A\), and \(R\) is the total number of cases. The larger \(DI\), the better the discrimination between cases belonging and not belonging to \(A\). Notice that for a perfect classifier with outputs of only zero and one, \(M = 1\), \(N = 0\), \(V = 0\), and \(W = 0\), making \(DI\) infinite.

Nonclassification, Supervised Learning Problems

With two exceptions, nonclassification, supervised learning problems constitute any continuous valued output, supervised learning problem other than classification. The exceptions are heteroassociative and autoassociative binary output problems such as mapping, data compression, and dimension reduction.

We consider the continuous output value case first. PMs for these problems are usually a function of a residual or error term \((O_i - D_i)\), where \(O_i\) is the observed value of an output PE for the \(i\)th case and \(D_i\) is the desired value of the same output PE for the \(i\)th case. Observed values are actual output PE values generated by the ANN. Desired values are correct values (i.e., values the ANN is trying to learn) and are sometimes called target values. Be warned that different authors use the same terminology to mean different things; some use “observed
value" to mean the actual value of an output PE, whereas others use it to mean the correct value.

Common PMs include (1) the average (or mean) absolute error (or deviation), $\sum |O_i - D_i|/N$, where the sum is over $i$ and $N$ is the number of cases; (2) the average (or mean) squared error (sometimes called PRESS or $SEC^2$), $\sum (O_i - D_i)^2/N$; (3) the root-mean-square error (RMSE), which most authors take as $\left[\sum (O_i - D_i)^2/N\right]^{1/2}$, but which others take as $\left[\sum (O_i - D_i)^2\right]^{1/2}/N$; and (4) the Pearson product–moment correlation coefficient, or simply the correlation coefficient. This coefficient is defined as follows:

$$r = \frac{\Sigma (O_i - O_{av})(D_i - D_{av})}{\sqrt{\Sigma (O_i - O_{av})^2 \Sigma (D_i - D_{av})^2}}$$

[45]

where the subscript "av" denotes an average. It can take values between $-1$ and $+1$. Values of $\pm 1$ indicate perfect positive or negative correlation between observed and predicted values, and a value of zero indicates no correlation. Be warned that perfect positive correlation does not necessarily mean perfect agreement. If agreement between observed and predicted values is perfect, then a plot of one versus the other should give a straight line whose equation is $O = D$. It is possible that a plot could give a line whose equation is $O = mD + b$, with $m \neq 1$ and $b \neq 0$. In this case, the correlation could be 1, but the agreement would not be perfect.

Other related PMs may be found in Ref. 114. Unfortunately, there is usually more than one name for a given PM, and worse, there is sometimes actually more than one PM for a given name (RMSE is an example). Authors would do a great service to readers by mathematically defining the PMs they use. These PMs can be applied to one output PE at a time or to all output PEs (i.e., globally). As stated, we recommend applying them to one PE at a time, since a global application may mask poorly performing PEs just as in the classification case.

As with the PMs discussed for classification problems, the PMs here can be averaged over a set of jackknife calculations and also used to determine a stopping point for training. Many software packages let you monitor the RMSE as training progresses and also let you stop training at a prescribed value of RMSE. But remember, a minimum RMSE for a training set does not guarantee a minimum RMSE for a test set. Also, these packages sometimes take $N$ in the denominator of the definition of RMSE to be the number of output PEs, not the number of cases. Make sure you know what the program is calculating.

A PM that is often used in jackknife calculations is the cross-validated correlation coefficient denoted typically by $cv_r^2$ or $q^2$. For one output PE, it is given by

$$1 - \frac{[\Sigma (O_i - D_i)^2/\Sigma (D_i - D_{av})^2]}{V} = (V - \text{PRESS})/V$$

[46]
where $V$ is the variance of the desired values with respect to the average of the desired values, that is, $V = \Sigma (D_i - D_{av})^2/N$. The term in square brackets in Eq. [46] is a normalized error for the given output PE.62

Finally, a few brief comments regarding the exceptional heteroassociative and autoassociative binary output problems. Although these are not classification problems, we feel that classification PMs are appropriate to apply here because the outputs are either zero or one. In most cases you should probably apply these PMs globally if you are interested in the compression, reduction, or mapping of an entire data set. On occasion, however, a few of the output PEs may not perform well, degrading the quality of the compression, and so on; PMs applied to individual output PEs may be helpful in such cases.

Miscellaneous Remarks

There is a tendency for PMs to be discipline specific. Physicists, for example, often use PMs based on information theoretic and entropy concepts. These metrics appear to have seen limited use in chemistry, and the interested reader is referred to Ref. 246 for an application example, and Chapters 5 and 6 of Ref. 13 for a theoretical discussion.

Receiver operating characteristic (ROC) curves were originally used in psychology and in evaluating electronic communications systems. Within the last decade they have been used in medical diagnosis systems. An ROC curve is constructed by plotting the true positive ratio versus the false positive ratio for a variety of threshold values. The larger the area under the curve, the better the performance of the ANN. The shape and area may also help you decide what threshold value is best for an application and indicate how ANN performance changes with any adjustable parameters in the network (see Refs. 248–250 for more information). The real point of the last two paragraphs is that many PMs can be used, but to find them, you may need to look outside the chemical literature.

**ANALYSIS OF NEURAL NETWORKS**

We said previously that a shortcoming of ANNs is the difficulty of determining how a network makes its decisions. While this is a difficult task, it is certainly not impossible to solve, and now we turn to a description of some ways of accomplishing this.

In general, if the weight between a given input PE and another PE has a relatively large magnitude, then the corresponding input variable may be important to how the network learns. However, the distribution of actual input values (the values going into the network after any preprocessing of the raw data) may affect this general observation. If the magnitudes of the inputs are relatively...
large, then large weights are likely to be important, but if the inputs are small, then large weights are probably less important.

To go beyond this, a deeper level of analysis is required. One of the first attempts at analyzing weights was that of Gorman and Sejnowski,\textsuperscript{44} whose method we briefly describe here (consult the reference for more details). These investigators used a backpropagation network (the method should be applicable to a few other ANNs as well) to classify sonar signals according to objects from which they were reflected. From 208 sonar signals (i.e., input patterns or state vectors) and the vector of weights between input and hidden layer PEs, the authors constructed 208 weight–state vectors. The distance between each pair of these vectors was calculated, and then a hierarchical clustering method was applied. Each cluster contained weight–state vectors that were similar to each other. The centroid of each cluster was computed and then passed through the trained network, after which the centroids were ordered according to the output each produced from a chosen hidden layer PE. Each centroid corresponds to a sonar signal. It was found that the weight vector was encoding (i.e., learning) three features of the sonar signals: bandwidth, onset, and decay.

Another approach to weight analysis is to consider just the weights as forming a vector. Consider a three-layer network (input, hidden, and output). If all the network weights are considered to be components of a vector $W$, then this vector will be of dimension $N = (n_i n_h + n_h n_o)$, where $n_i$, $n_h$, and $n_o$ are the numbers of input, hidden, and output PEs. [If there is a bias connected to the hidden and output PEs, $N = (n_i + 1) n_h + (n_h + 1) n_o.$] If we define an $N$-dimensional unit vector as $S = N^{1/2}(1, 1, \ldots, 1)$, then the angle $W$ makes with $S$ is

$$\theta = \arccos[(W \cdot S)/||W||]$$

where $||W||$ is the length of $W$. Plotting $||W||$, $\theta$, and one or more PMs against the number of training epochs typically will reveal that some combination of these quantities changes relatively rapidly over some portion of training. This is usually an indication that the network is in the process of learning decision rules.

For example, during the first 100 epochs you may find that $||W||$ and percent correct are essentially constant but $\theta$ decreases quite rapidly. It is likely that during this stage of training the ANN is learning some useful rule even though percent correct is not changing; apparently the weight vector is adapting to the correct orientation in weight space. This conclusion would be corroborated if, during further training, $\theta$ remained constant or continued to decrease. If $\theta$ began to increase after it has been decreasing, you might conclude that the network began to learn an incorrect rule, "realized" this, and adapted, hopefully to learn a correct rule. During the next 100 epochs, you may find that $\theta$ is constant, but both $||W||$ and percent correct increase. This would be an indication that the ANN is learning a second rule. Of course, the same plots could be
made using only weights between the input and hidden layers (or between the hidden and output layers) as elements of the weight vector. You would lose some of the global information about rule learning by the network, but the plots might be easier to interpret. These plots will not tell you what rules the network is learning, but they can give an indication of how many rules to look for.

To ascertain what rules an ANN learns, one can resort to some form of sensitivity analysis. In its most general form, sensitivity analysis addresses the following question: For a given change in an input or inputs, what is the corresponding change in an output or outputs? Sensitivity analysis can be utilized in several contexts, one of which is pruning.

Pruning methods are used to eliminate redundant weights and to simplify network architecture by eliminating inter-PE connections having small weights. In some cases, all connections between an input PE and other PEs may be eliminated. This is of particular interest because it tells you that the input variable corresponding to that PE is apparently not relevant to the rules an ANN is learning. Sometimes after a pruning procedure, only input PEs that have relatively large weights to HL PEs will be left, making it possible to make strong statements about the importance of the input variables in determining outputs. Sensitivity methods of pruning define a “sensitivity” that is some function of weights. Different methods use different functions and also different (and somewhat arbitrary) criteria for the elimination of PEs. Generally, PEs with the smallest sensitivities are eliminated, and these PEs also tend to have the smallest weights. These procedures are somewhat analogous to stepwise regression, where statistical tests are used to determine which subset of input variables consists of statistically significant contributors to the “best” regression curve. In pruning methods, one hopes to also obtain the best subset of input variables, although usually statistical tests are not used. An exception to this may be found in Ref. 252, where a stepwise approach to variable elimination using the F ratio is taken. The elimination of variables is stopped when the F ratio for all variables is greater than some predetermined number. Since the outputs of the ANNs in that study were not normally distributed, the F ratio was not used for significance testing. Descriptions of different methods (some of which are network-type specific) may be found in Refs. 253–261. Applications of several pruning methods to the problem of variable selection may be found in Refs. 262–267.

Sensitivity analysis may also be used outside the context of variable reduction. Kocjancic and Zupan used a backpropagation network (the method described should also work with other networks) with two HL PEs to map data to two dimensions by plotting the outputs of the hidden PEs against each other. Well-defined and separable categories were formed. For a given category, all but one of the input variables were held at their average values. The other variable was varied from its minimum value to its maximum. Passing the resulting patterns through the network resulted in a curve when the corresponding outputs of the hidden PEs were plotted. By noting whether this curve remained...
completely inside the category region, the authors were able to make qualitative statements about the effect of the variable in determining category membership.

Relation factors (RFs) indicate the strengths of relationships between individual input PEs and individual output PEs.\textsuperscript{242} There are two types of RF: RF1 and RF2. RF1 is the effect of a given input on a given output when all other inputs are zero. It is calculated by subtracting the value of the given output PE with all inputs equal to zero from the value of the same output PE when the value of the given input PE is one and all other inputs are zero. There are \( n_p n_o \) RF1s for a given network. Positive values of an RF1 indicate an excitatory effect of the input PE on the output PE; negative values indicate an inhibitory effect, and the greater the magnitude of RF1, the greater the magnitude of the effect.

RF2 is slightly different in that it measures the effect of a given input on a given output averaged over a set of input patterns. RF2 values give useful information about a particular data set but, unlike RF1 values, they are not transferable between data sets. RF2 values are calculated as follows: for the first case in a data set, determine the output value of the given output PE when the input to the given input PE is equal to zero and all other inputs are equal to their “case one” values. Subtract this output value of the given output PE from the output value of the same output PE when the input to the given input PE is equal to one and all other inputs are still equal to their case one values. Repeat this calculation for each case in the data set. Adding all the differences in output PE values and dividing by the number of cases gives RF2 for the given input PE–output PE pair. The entire calculation is repeated for each possible input PE–output PE pair, giving \( n_p n_o \) RF2 values.

Aoyama and Ichikawa\textsuperscript{268} have given analytic formulas for the partial derivatives of the output value of either a HL or output layer PE with respect to the input value of an input PE. Their formulas, which are applicable to any feedforward network with differentiable transfer functions in the hidden and output layers, allow you to give a precise, analytical answer to the question that sensitivity analysis asks (see above). A similar sensitivity analysis has been performed for a radial basis function ANN.\textsuperscript{269} Aoyama and coworkers introduced another technique useful in network analysis: the reconstruction of weight matrices for a backpropagation network.\textsuperscript{107,270} They used a learning rule to train a backpropagation network and then erased some of the learning by essentially using an “unlearning” rule. This learning–unlearning procedure was done several times in the overall training of the network, with the unlearning rate set to about a tenth of the learning rate. The authors discovered that some weights are affected more strongly by learning than by unlearning (those weights being strengthened), while other weights are affected more strongly by unlearning, resulting in weakened or null connections. They studied several data sets and showed that this reconstruction learning made possible very clear interpretations of what the network learned. In particular, it was very easy to trace connections between input and output PEs and to understand the roles input variables played in determining network output.
Another approach to ANN analysis is to study PE outputs (also called activities, activations, or activity level). Lehky and Sejnowski\textsuperscript{271} used a back-propagation network to determine curvatures from images of simple geometrical surfaces. After training, they presented each of the 2000 training images to the network and generated histograms of the activity of each HI PE. These histograms formed two groups, one with unimodal and one with bimodal distributions. PEs with unimodal distributions were detecting orientation or amplitude, while PEs with bimodal distributions were making either/or decisions.

We end this section by considering two examples of network analysis that combine considerations of both weights and activities. Ruisanchez et al.\textsuperscript{180} classified energy dispersion X-ray spectra, using an unsupervised Kohonen network to form 12 clusters. The clusters were determined by the active PEs in the two-dimensional Kohonen layer when X-ray patterns were passed through the network. A two-dimensional cluster map was formed by noting the location of the active PEs for each pattern. Weight maps from individual inputs were superimposed on the cluster map. By simultaneously superimposing seven of the weight maps on the cluster map, it was possible to generate 12 rules for classifying spectra into the 12 categories. Using these 12 rules enabled all 95 spectra in a test set to be classified correctly. Domine et al.\textsuperscript{85} performed a very similar analysis using an ART network on a set of aliphatic substituents. Activity and weight maps were generated that allowed the authors to demonstrate that certain input variables discriminated between clusters. They did not publish any classification rules.

CONCLUDING REMARKS

In this chapter we have provided a tutorial on neural networks, showing you how they operate, how and when they should be used, and the many chemical applications to which they have been applied. Here we wish to make several comments and observations, indicating some possible avenues for further work.

Most, if not all, of the topics we have discussed are still the subjects of current neural network research, both in the chemical and neural network communities. There is a wealth of research being done by neural network theorists on topics that are relevant to chemists. To illustrate this, we perused four issues of the journal \textit{Neural Networks} from August 1998 to April 1999, and found at least 10 papers that should be of interest to chemists. We briefly review these papers because they illustrate some possible directions for future work. Perhaps more important, they are examples of an enormous number of methods and approaches that exist in the neural network community. We be-
lieve that chemists have been somewhat remiss in not applying more of these techniques to chemical problems.

Advances are being made even in ANNs that are historically "old." Two papers have presented variations on the BAM learning rule, which resulted in significantly larger storage capacity and more efficient pattern retrieval in BAM networks. A variation in the perceptron learning rule resulted in a network that is guaranteed to correctly classify linearly separable two-category data and to indicate conclusively when data is not linearly separable. This is a significant achievement because in most real-world problems it is difficult if not impossible to have a priori knowledge of whether data is linearly separable. Also, a recursive deterministic perceptron ANN has been developed that is capable of solving any two-class classification problem. The standard BP ANN has been modified to avoid false local minima. In trial computations with this modified network, global minima or near-optimal solutions were obtained. The modification can also be used with the DBD BP network. Another variation on the standard BP network has been developed that gives superior results for noisy data. A variation of an RBF network has been used for finding nonlinear principal components. A linear eigenspace separation transform has been derived that allows the reduction in size of a neural network while enhancing its generalization accuracy as a binary classifier. Significantly, classification was excellent even when the original data distributions were nonnormal, had equal class means, and had unknown a priori probabilities for the two classes.

A new approach to dealing with overfitting and the selection of optimal or near-optimal network architecture without the use of a validation or test set has been developed. The approach used optimal brain damage pruning and enabled a test set error estimate to be made. Statistical methods (hypothesis testing, etc.) have been used to guide selection of optimal network architectures. Finally, templates have been used for extracting various types of symbolic rules from networks that use sigmoidal transfer functions. These authors also give a good review of various other methods that have been used in the analysis of ANNs.

The Neural Networks papers we have cited are just a very few examples of the existing research on neural networks that could potentially reap rewards for chemists. We also noted earlier in the chapter that neural networks of other types warrant more attention from chemists; these include GRN, RBF, PNN, and recirculation neural networks.

It is clear that artificial neural networks have become a viable tool for chemists. It is not clear, however, that they are consistently superior to statistical methods. It does seem obvious that with a few possible exceptions (counter-propagation and Kohonen networks), it is more difficult to derive physical meaning from network analysis than it is from statistical methods. Even relatively simple and straightforward questions (e.g., Which variables are most
important in determining a property of a system?) are usually more easily answered from statistical methods than from ANNs. This situation is likely to continue until more research in network analysis is done.

What does not seem so obvious is whether ANNs consistently give better prediction results than statistical methods. In the early to mid-1990s a relatively large number of papers were published claiming superior results from ANNs, usually in comparison to linear statistical methods, but sometimes to nonlinear ones as well. In the last few years, more published studies seem to be casting doubt on this conclusion. Unfortunately, many of these comparative studies contrasted only one or two network types with one or two statistical methods, and many were carried out on data sets that were not terribly well characterized or understood. It would be very useful to find several well-understood data sets for each different problem type (classification, modeling, etc.) and apply as many applicable statistical and ANN methods to them as possible. These studies should use several well-defined performance metrics, and care should be taken to optimize all network parameters. Most importantly, every published report of these studies should give enough explicit detail of all aspects of all performed computations (e.g., precisely how the number of hidden nodes was determined, exactly how the data was preprocessed, what specifically was done to optimize learning parameters, and so on) to render the results truly reproducible by anyone reading the report. Studies of these types would provide an invaluable "database" from which a clear picture of both the capabilities and limitations of ANN versus statistical methods would hopefully emerge.

**APPENDIX: NEURAL NETWORK SOFTWARE**

Over the past few years, the number of freeware, shareware, and commercial neural network software packages has increased dramatically. These packages range from highly specialized programs designed for a particular kind of data analysis and containing only one or two neural network types to comprehensive programs containing 20–30 network types and designed for general-purpose analysis. Rather than attempt a comprehensive listing and description of different packages, we simply give some Internet sites which, taken together, form a good starting point for acquiring software.

As with any resource on the World Wide Web, new websites related to neural networks are coming into existence and others are disappearing. When the proofs for this chapter were being checked and revised, the Universal Resource Locations (URLs) for this appendix were tested. Connections could not be made to more than one-third of the URLs in our original list. Only URLs that
worked as of May 2000 are presented here. A Web search engine such as Altavista will turn up many, many hits.

Battelle Pacific Northwest National Laboratory maintains three websites that are perhaps the most comprehensive sites available. The site at (all addresses are preceded by http://)

www.emsl.pnl.gov:2080/proj/neuron/neural/systems/software.html contains brief descriptions of about 60 commercial neural network programs, whereas

www.emsl.pnl.gov:2080/proj/neuron/neural/systems/shareware.html contains similar information for about 60 shareware packages. At both sites the description consists of platforms, program options, and supplier contact information (usually address, phone, fax, e-mail, and web address). The third site

www.emsl.pnl.gov:2080/proj/neuron/neural/gateway/USA.html contains much the same information as the two sites above but also has links to many academic research groups.

Other websites that contain links to some software as well as links to a wide variety of neural network resources are:

www-sci.sci.kun.nl/cac/www-neural-links.html
clareet.psychology.mcmaster.ca/NeuralComp/companies.html
Alston University Neural Computing Research Group has downloadable software available at

www.ncrg.aston.ac.uk/Welcome.html

Information about Andrew Hunter's Sunderland University Genetic Algorithm package is at

www.trajan-software.demon.co.uk/sugal.htm

The University of Nevada, Reno distributes NevProp4 (ANN statistical prediction software) at

www.scs.unr.edu/nevprop/

Many commercial distributors have free downloadable 30-day trial periods for at least some of their products. Sometimes this is merely demonstration software, but more often than not, you have the opportunity to test-run a full software package. This is an invaluable experience and will provide you with much more information about a particular package than we could possibly provide here. You can learn a great deal about neural networks in general and also which packages are best suited to your particular data analysis problems (not to mention your particular personal preferences).
PRONET, a program to predict protein backbone conformation. Centro di Ricerca, Sviluppo Studi Superiori in Sardegna, Italy.

www.crs4.it

Perceptron Simulator for Drug Design (PSDD)
Quantum Chemistry Program Exchange (QCPE), Indiana University, Bloomington

cqpe.chem.indiana.edu/catalog.html


www.oxmol.com

REFERENCES

References


132 Artificial Neural Networks and Their Use in Chemistry


References


136 Artificial Neural Networks and Their Use in Chemistry


Artificial Neural Networks and Their Use in Chemistry

138

212. PV-WAVE, Visual Numerics Inc., 9990 Richmond Avenue, Suite 400, Houston, TX 77042; tel. 800-364-8880; Web www.vni.com.


References


