Chapter 18
Artificial Neural Networks

Abstract Neural network models are intriguing because they are based on the intuitive notion of mimicking the structure of neurons that constitute the human brain. More importantly to database marketers, neural networks can provide great flexibility in handling non-linearities and variable-interactions that can be important in predictive modeling applications. We describe the neural net model, how it is estimated, and more advanced forms of neural networks.

18.1 Introduction

Artificial neural networks (ANN) are one of the most powerful tools in data mining. Based on a model analogous to the human brain, ANNs "learn" and generalize from external inputs. When exposed to "training observations", neural networks discover patterns and relationships. Their approach is fundamentally different from the way that traditional digital computers solve problems. Taking a top-down approach, computers solve problems as commanded by a series of instructions supplied by humans. In contrast, ANNs take a bottom-up approach, learning from examples and searching for patterns to infer important relationships.

18.1.1 Historical Remarks

Artificial neural networks have a long history. The basic idea dates to McCulloch and Pitts (1943) who developed a model to explain how biological neurons work. McCulloch was a neuroscientist and Pitts was a mathematician. Combining studies in neurophysiology and mathematical logic, they attempted to explain how the human brain works and described the logical calculus of neural networks (Berry and Linoff 1997; Haykin 1999).

Considered the founders of neural networks and artificial intelligence, McCulloch and Pitts demonstrated that a network with a sufficient number of neurons and synaptic connections could compute any computable function. Hence, everything that can be done with a computer can be done with a neural network. Their model provided a new approach to solving various decision-making problems even though their original goal was to study how human brains worked.

There were not many applications of neural networks by the 1980s, partly because of limited computing power and a theoretical deficiency of the original neural network model noted by Minsky and Papert (1969). The recent growth in applying neural networks is generally agreed to be due to the paper by Hopfield (1982) and the book edited by Rumelhart and McClelland (1986). Using ideas from statistical physics, Hopfield overcame the theoretical weaknesses of McCulloch and Pitt's model. In addition, the well-known book, *Parallel Distributed Processing*, edited by Rumelhart and McClelland popularized the back-propagation algorithm that made neural networks practical.

18.1.2 ANN Applications in Database Marketing

The area of artificial neural networks is multidisciplinary. Researchers from neurophysiology, statistics, mathematics, computer science and engineering have contributed to the development of its concept and methodology. Accordingly, neural networks find applications in various fields including pattern recognition, signal processing and control, speech recognition, fraud detection, demand forecasting, and so on. By the 1990s, there were more than one hundred applications to business problems alone (Sharda 1994; Wong et al. 1995)

Since artificial neural networks comprise a class of general-purpose tools, there are many published applications to marketing. ANNs were compared to traditional econometric models in forecasting aggregate market demand (e.g., Hruschka 1993; Gruca et al. 1999). ANNs were also applied to market segmentation (Fish et al. 1995; Balakrishnan et al. 1996; Hruschka and Natter 1999), target mailing (Zahavi and Levin 1997) and other marketing problems (Yao et al. 1998; Knott et al. 2002; Kim et al. 2005).

ANNs can be applied to classification, prediction and clustering tasks that are all about what database marketing modelers do. For example, Balakrishnan et al. (1996) applied neural nets to market segmentation. The data employed in their study represent the brand switching probabilities on 18 different coffee brands for each of 207 households. They employed the neural net to cluster these 207 households represented by a vector of switching probabilities for 18 brands of coffee. Similarly, Hruschka and Natter (1999) employed neural networks to cluster 831 housewives. Each housewife was distinguished in her usage of household cleaner brands, demographic characteristics

(age, household size, number of children, housewife's education, etc.) and her attitude variables (e.g., cleaning the household is cumbersome).

ANNs have also been applied to find target customers. Zahavi and Levin (1997) explored the feasibility of using neural networks as a means for targeting audiences for promotion through the mail, from a house list. Knott et al. (2002) applied neural nets to predict which product a customer was most likely to buy next, given the set of products the customer already owned. They applied the approach to a retail bank trying to identify customers who would be receptive to a certain type of loan. They found in a field test that the approach generated more profits than the heuristic approach the company was currently using. On the other hand, Kim et al. (2005) proposed an approach to employ ANNs guided by genetic algorithms for targeting households. They applied their procedure to a solicitation of 9,822 European households to buy insurance for recreational vehicles. Their model performed better than traditional logistic regression (with a principal component analysis) in targeting households interested in purchasing the insurance policy.

18.1.3 Strengths and Weaknesses

There are several driving forces contributing to the wide applications of ANNs for the last 2 decades. The broad availability of high-speed computing allows a sophisticated model like ANN to be handled within a reasonable amount of time. In addition, practitioners without strong statistical knowledge can implement ANNs due to the availability of off-the-shelf neural nets software.[1] More importantly, neural nets are no longer treated as black boxes. Statisticians have shown that ANNs are highly nonlinear regression models and a number of traditional statistical models such as linear and logistic regression are special cases of ANNs (White 1989, 1992). As a result, practitioners can be comfortable in employing ANNs because we know they are closely related to traditional statistical models.

Some commercial software vendors often mislead users by exaggerating the automatic features of their products. However, like the application of formal statistical techniques, successful applications of ANNs require deep understanding of neural net theory and their application domains. The user's subjective judgments will get involved in determining the network architecture and training parameters. Special care should be taken to check the

[1] Various neural network routines (e.g., multilayer perceptron and radial basis function) are available in SAS Enterprise Miner. Neural Connection from SPSS also delivers all the tools of neural network modeling for prediction, classification and time-series analysis. Advanced Software Applications (ASA) provides software called "ModelMax" particularly adapted to database marketing prediction applications.

performance of neural nets in the validation sample since ANN tends to overfit the training samples (see Chapter 11).

Several researchers compared ANNs with established statistical techniques such as clustering, logistic regression, discriminant analysis, time-series methods, decision trees etc. and found ANNs to be superior. For example, Fish et al. (1995) compared neural nets with discriminant analysis and logistic regression for industrial market segmentation. Neural nets were found to achieve higher hit ratios on the holdout sample than the other statistical techniques. Hruschka and Natter (1999) compared the clustering performance of ANNs to the K-means clustering technique and found that ANNs were better.[2] Time series forecasts produced by neural nets were compared with forecasts from six statistical time series methods (Hill et al. 1996). Across monthly and quarterly time series, the neural nets did significantly better than traditional time series methods. ANNs were particularly effective for discontinuous time series data.

On the other hand, other researchers have not been able to show the superiority of ANNs over traditional statistical techniques. For example, Zahavi and Levin (1997) compared neural nets with logistic regression for targeting customers for promotional mailing offers. Their results showed that the fit achieved for both methods was approximately the same but the interpretation was easier for logistic regression. Brown et al. (1993) compared backpropagation neural networks with decision trees on three problems that are known to be multi-modal. Their results suggested that there was not much difference between both methods. Comparing ANN (multilayer perceptrons) with decision trees (CART), Atlas et al. (1990) also found that there was not much difference in accuracy. Balakrishnan et al. (1996) compared neural nets with K-means algorithm and found that there was not much difference between two methods. However, a combination of the two methodologies, wherein the results of the neural nets are input as seeds to the K-means, provided more insightful segmentation schemes. More recently, Linder et al. (2004) compared neural nets with decision trees and logistic regression in simulated direct marketing data. ANNs outperformed the other two methods when the sample size was small, but decision trees and logistic regression yielded better results when sample size was large – with logistic regression being generally superior to decision trees. These results are rather surprising, because in other research simple models tend to outperform more complex models with small number of training examples or highly noisy data (Hastie et al. 2001).

Summarizing, we conclude that the relative performance of ANNs to traditional statistical methods depends on the types of data and applications. The main strength of ANNs lies in their ability to model highly nonlinear relationships and interactions with few a priori assumptions specified. Hence,

[2] Their neural net model is not the self-organizing maps described in Clustering chapter. They constructed a feed-forward artificial neural network specially designed in solving the problem of cluster-based market segmentation.

ANNs have been shown to outperform traditional statistical techniques when there exist highly nonlinear relationships and/or when there are significant interactions among independent variables (Rumelhart and McClelland 1986; Hill et al. 1996). If the true relationship is linear (or logistic), the linear (or logistic) regression will work better than an unnecessarily complex neural network because its linearity (or logistic) assumption functions like additional prior information. On the other hand, when the true relationship is complex, ANNs may outperform linear (or logistic) regression because the wrong assumption made in linear (logistic) regression will bias its result. More extensive research is required to find the areas of marketing problems in which ANNs can have unique advantages. From a practical standpoint, neural nets have certainly proven that belong in the "consideration set" of database marketers, especially under the conditions mentioned above. Our recommendation is the same as with the potential application of any statistical technique – apply the neural net and the incumbent method to the same data, and see which does better on the holdout data.

18.2 Models of Neurons

In this section we present a general model of an artificial neural network. In Sect. 18.3, we will focus on the most commonly used specific form, the multilayer perceptron. In Sect. 18.4, we describe another specific form, the radial-basis function network, which is currently less popular than the multilayer perceptron but has much potential.

Artificial neural networks are composed of basic units designed to mimic the behavior of biological neurons. ANN's are analogous to an organism's nervous system: stimuli or "inputs," if they are strong enough, cause neurons to fire off, in turn causing the organism to respond. In statistical terms, the inputs are the independent variables and response is the output or dependent variable. Neural nets can handle both categorical and continuous data, both for independent and dependent variables.[3] Output variables can involve just one response, or more than one (e.g., whether the customer will respond to a catalog, and if so, how much will the customer spend). As shown in Fig. 18.1, a neuron is an information-processing unit that translates input signals into

[3] Categorical data can be handled in two ways. The first is to treat each categorical feature as discrete, ordered value. For example, we assign 0.0 for brand A, 0.5 for brand B, and 1.0 for brand C for the brand choice variable with three brands. This method is somewhat problematic since the neural net will assume that the codes are ordered (i.e., brand A and C is far apart). The second way of handling categorical features, which is more popular, is to represent the categories by a set of dummy variables. For example, we create brand A, brand B and brand C dummies for the three-brand choice variable. Each brand dummy will take the value of 1.0 if the brand is chosen, and 0.0 otherwise. And for identification, one of the dummy variables is dropped before estimation.

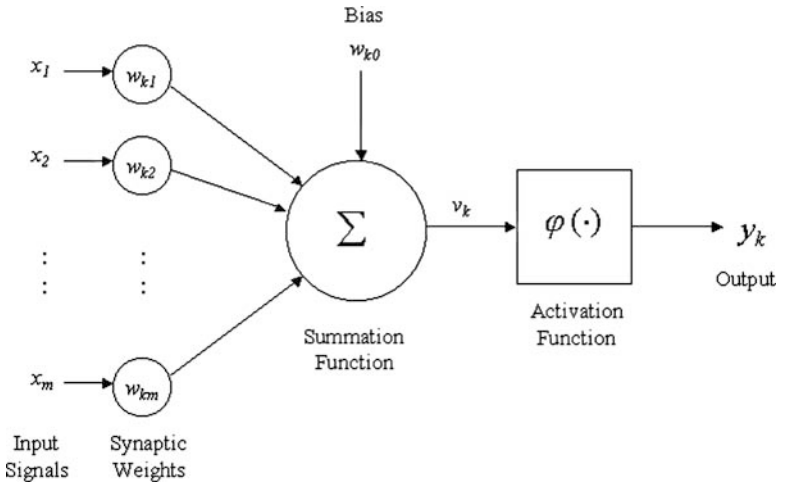

Fig. 18.1 A model of a neuron (adapted from Haykin, Simon, Neural Networks: A Comprehensive Foundation, 2nd Edition, (c) 1999, pg. 11. Reprinted by permission of Pearson Education, Inc., Upper Saddle River, NJ.).

outputs. There are three basic elements of any neural network model: synaptic weights, the summation function and the activation function.

Input x_j is connected to neuron k by a synaptic weight w_{kj}. Assigning different weights on each input implies that the importance or the strength of each input is different in producing an output. Input values with their weights are combined by a summation or combination function $\sum$. The most popular function is the linear combination function that can be written as

$$v_k = \sum_{j=0}^{m} w_{kj} x_j = w_{k0} x_0 + u_k \tag{18.1}$$

where x_0 is set to be one. That is, the linear combination function is the weighted sum of all input values, where each weight is given by its synaptic weight. Note also that the above linear combination function contains a term $x_0 = 1$ with its associated weight of w_{k0}. The term w_{k0} plays a role of applying an affine transformation to the value of u_k (Haykin 1999). It has the effect of increasing or decreasing the net input of the activation function (u_k), depending on whether w_{k0} is positive or negative, respectively. In other words, its role is similar to that of the intercept term in linear regression.

Another important element in the neuronal model is the activation function that transforms the value from the combination function into the output. That is, $y_k = \varphi(v_k)$ where y_k is the output of the neuron k and $\varphi(\cdot)$ is the activation function. Its major role is to limit the amplitude of the output of a neuron. Typically, the range of the normalized amplitude for the output of a neuron is written as the closed unit interval [0, 1] or [-1, 1]. There are several types of activation functions: the linear, threshold, piecewise-linear, logistic

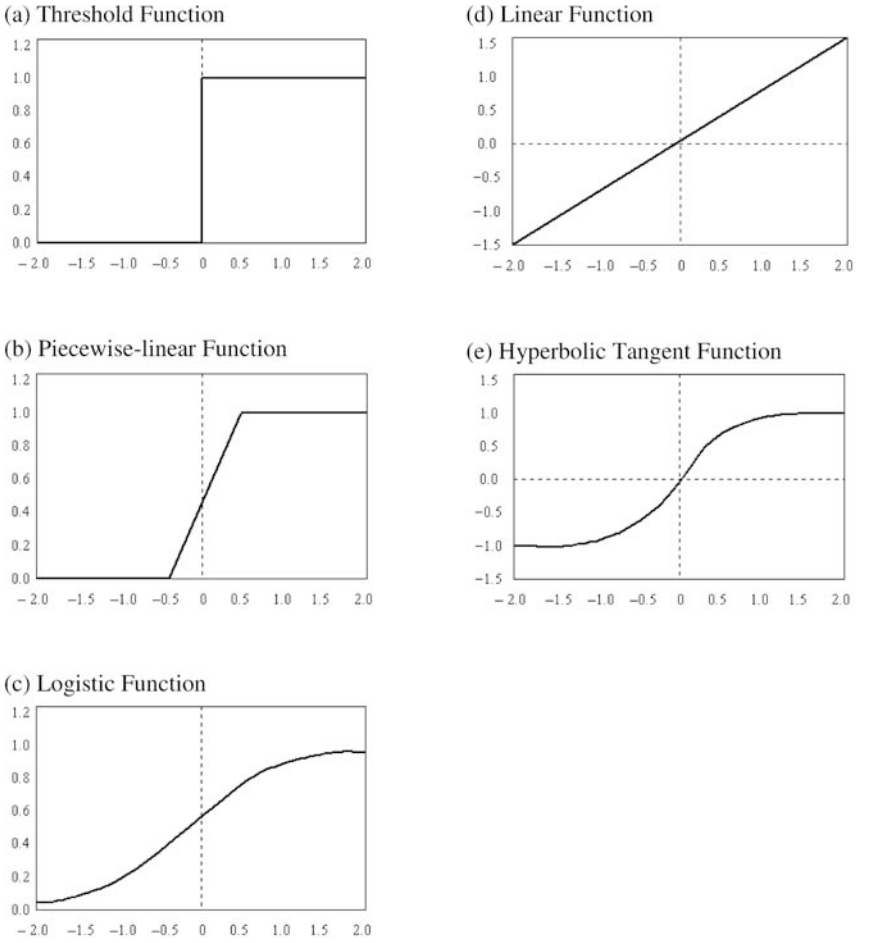

Fig. 18.2 Five types of activation functions.

and hyperbolic tangent (Berry and Linoff 1997; Haykin 1999). Figure 18.2 shows the shapes of these five activation functions.

For the threshold activation function, the output unit takes on the value of one if the value of v_k is nonnegative, and zero otherwise (see Fig. 18.2(**a**)). That is,

$$\varphi(v_k) = \begin{cases} 1 & \text{if } v_k \geq 0 \\ 0 & \text{if } v_k < 0 \end{cases} \tag{18.2}$$

The model by McCulloch and Pitts (1943) employed this threshold function. And the threshold function is commonly called as a Heaviside function (Haykin 1999).

Figure 18.2(**b**) shows the shape of the piecewise linear function that can algebraically be written as

$$\varphi(v_k) = \begin{cases} 1 & \text{if } v_k \geq 0.5 \\ v_k & \text{if } -0.5 < v_k < 0.5 \\ 0 & \text{if } v_k \leq -0.5 \end{cases} \tag{18.3}$$

The piecewise linear function approximates the nonlinear logistic function in a linear form. The choice of the lower and the upper limit for the linear region (e.g., -0.5 and 0.5 in Fig. 18.2(**b**)) is up to the user. The piecewise linear function generalizes both the threshold and linear activation functions. It becomes the threshold function if we make the upper limit to be very close to the lower limit so that the linear region vanishes. Alternatively, if we do not specify the upper and lower saturation ranges, it will become the linear function shown in Fig. 18.2(**d**).

The S-shaped logistic function is the most frequently used activation function in ANNs. Its functional form shown in Fig. 18.2(**c**) is defined as

$$\varphi(v_k) = \frac{1}{1 + \exp(-\alpha v_k)} \tag{18.4}$$

where α is the slope parameter of the logistic function. The logistic function exhibits linear behavior when the absolute value of v_k is small. However, as it gets larger, the logistic function gradually approaches either 0 or 1. This property of gradual saturation is a reasonable one in modeling various social (and natural) phenomena. Moreover, different from the piecewise linear, the logistic has an attractive mathematical property of differentiability.

Finally, the hyperbolic tangent function is different from the logistic function in that its range of the activation function is from -1 to 1 instead of 0 to 1. Shown in Fig. 18.2(**e**), the hyperbolic tangent is S-shaped similar to the logistic function, but the lower saturation is negative one rather than zero. The hyperbolic tangent function is defined as

$$\varphi(v_k) = a \tanh(b v_k) \tag{18.5}$$

where a and b are the parameters controlling the shape of the hyperbolic tangent. With suitable values for a and b (e.g., $a = 1.7159$ and $b = 2/3$), the activation function becomes anti-symmetric, that is $\varphi(-v_k) = -\varphi(v_k)$. The logistic function does not have this property. ANNs can learn faster when the activation function is anti-symmetric (LeGun et al. 1991).

18.3 Multilayer Perceptrons

In this section we study the most commonly used class of artificial neural networks, multilayer feed-forward networks. This network is especially useful

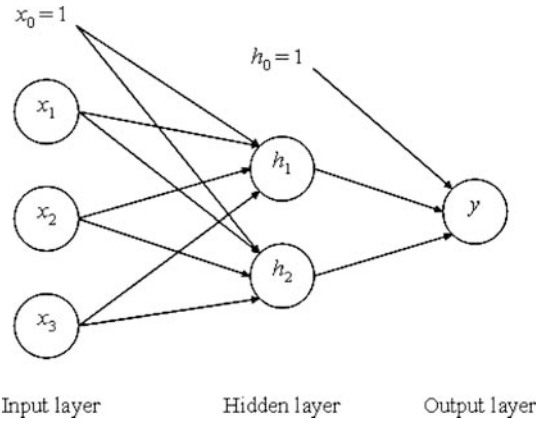

Fig. 18.3 Multilayer feed-forward network.

for prediction and classification tasks among others. We start our discussion with the "topology" or "architecture" of the multilayer perceptron, followed by its method of training, and the back-propagation algorithm. We then discuss more advanced issues in multilayer perceptrons: how to determine the number of neurons in the hidden layer, and the optimal value for learning rate and momentum parameters. This section also describes issues on transformation of input values and model validation.

18.3.1 Network Architecture

There are a number of different ways to classify network architectures. A primary differentiator is single-layer networks versus multilayer networks. Single-layer networks only consist of an input layer and an output layer, whereas multilayer networks have one or more hidden layers, as shown in Fig. 18.3. The networks can also be classified by the presence of at least one feedback loop. The networks without any feedback loop are called feed-forward networks where there is only one-way flow from input units to output units. Otherwise, they are called recurrent networks.

Figure 18.4 show an example of a recurrent network in which there are two inputs, two hidden neurons and an output. One feedback connection originates from the output and two feedbacks come out of hidden neurons. The presence of feedback loops can explain different learning capability of the network. For example, the feedback loops shown in Fig. 18.4 have the unit-delay term z^{-1} which incorporates dynamic learning behavior. The recurrent networks have typically been applied to (dynamic) time-series models such as adaptive equalization of communication channels, speech processing, plant control, and automobile engine diagnostics (Haykin 1999). Recurrent

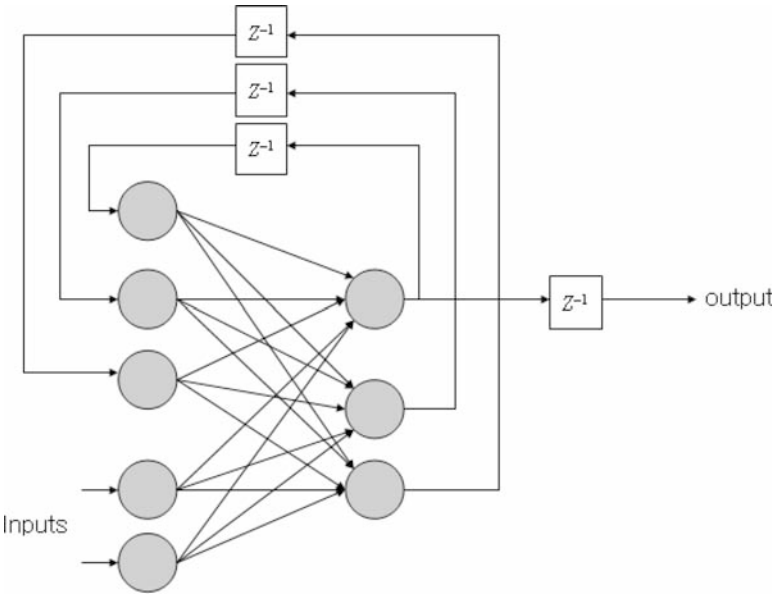

Fig. 18.4 Recurrent networks (From Haykin 1999).

networks have not been applied to database marketing problems, possibly because fewer dynamic models have been employed in database marketing applications. However, there are potential applications in areas such as multi-campaign management (see Chapter 28).

Returning now to the more commonly applied feed-forward multilayer neural network, also called the multilayer perceptron, Fig. 18.3 shows a typical structure. It consists of an input layer, one or more hidden layers, and an output layer. The multilayer perceptron in Fig. 18.3 has three input values (or independent variables in traditional regression models), one hidden layer, and one output value (or dependent variable). The hidden layer has two neurons. Hence, the multilayer perceptron in Fig. 18.3 is referred to as a 3-2-1 network. Finally, we say that the network is fully connected in that every node in each layer of the network is connected to every other node in the next or forward layer.

Positioned between the input and output layers, hidden layers translate the independent variables into a prediction of the dependent variables. Hidden layers are required for the ANN to model complicated interactions among input variables and other nonlinear relationships (Haykin 1999). As described above, a single-layer perceptron does not have hidden layers, but directly connects the input to the output layer.

Theoretically, a network can have any number of hidden layers. A network with a large number of hidden layers may be able to capture highly complicated relationships between inputs and outputs. However, a network with one hidden layer is frequently used for at least two reasons. First, a network

with a large number of hidden layers may over fit the data and just capture random noise in the data. We want to make our model as simple as possible to avoid the problem of overfitting. More importantly, researchers employ the network with one hidden layer without hesitation because of the "universal approximation theorem". The theorem roughly states that a single hidden layer is sufficient for a multilayer perceptron to approximate any continuous relationship between inputs and outputs (Barron 1993; Haykin 1999).[4]

We now write the neural network in algebraic form for the discussion of back-propagation algorithm in the next section. To avoid notational complexity, we limit our discussion to the multilayer perceptron example in Fig. 18.3. Three input signals come into the input layer pass forward through the network, and produce an output signal at the end. Let a training sample of size N be denoted by $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{d})$. Three input vector $\mathbf{x}_1, \mathbf{x}_2$, and $\mathbf{x}_3$ represent the input layer and the output (response) vector $\mathbf{d}$ represents the output layer.

Neurons in the hidden layer perform two functions: combination and activation. Each neuron in the hidden layers receives the input values (x_1, x_2, x_3) from the input layer, applies the weighted summation and activation functions, and sends the resulting values (h_1, h_2) to the output layer. More specifically, neuron 1 in the hidden layer receives the input values (x_1, x_2, x_3) from the input layer and their associated weights are (w_{11}, w_{12}, w_{13}) respectively. Similarly, neuron 2 in the hidden layer receives the input values (x_1, x_2, x_3) from the input layer and their associated weights are (w_{21}, w_{22}, w_{23}) respectively. The combination function calculates the weighted sum of the input values. The resulting intermediate value $v_j (j = 1, 2)$ is written as

$$v_j = \sum_{i=0}^{3} w_{ji} x_i \tag{18.6}$$

where j labels the hidden layer neuron, $x_0 = 1$ and w_{j0} is intercept term. Applying the activation function, we have the output value for neuron j equal to $h_j = \varphi(v_j)$. For example, if a logistic activation function is applied, $h_j = 1/[1 + \exp(w_{j0} + w_{j1}x_1 + w_{j2}x_2 + w_{j3}x_3)]$. The resulting output values h_1 and h_2 become the input values to the output layer. Given a logistic activation function, the output value $y = \varphi(v_3) = 1/[1 + \exp(w_{30}h_0 + w_{31}h_1 + w_{32}h_2)]$ where $h_0 = 1, w_{30}$ is for intercept term, and ν_3 represents the combination function relating the hidden layer to the output layer.

An interpretation of the above algebra is that we have two neurons; each has a probability h_j of firing off. Whether a given neuron fires off depends on the inputs or stimuli (x's) and how influential they are (w's). If enough neurons fire off, we get a response, i.e., the output, the dependent variable changes. This occurs through the function φ, which in turn depends

[4] However, the theorem does not say that the multilayer perceptron with a single hidden layer is the best. Interestingly, more hidden layers is often easier to implement and reduces learning time.

on whether the hidden layer neurons fire off (h's) and if they do, how influential they are in determining the response (represented by the w's).

18.3.2 Back-Propagation Algorithm

Training the network involves the process of finding the optimal weights (w_{ji}, where j indexes a node (either a hidden neuron or an output) and i the inputs that go into that node) to attach to the input values from the preceding layer. The process of neural net training is similar to the process of estimating parameters in nonlinear regression. The network searches for the optimal weights such that the predicted output value from the output layer is as close to the corresponding actual output value as possible.

The most well-known method of finding optimal weights for the multilayer perceptron is the back-propagation algorithm, which consists of two passes. We start with the calibration sample, also called the "training set." In the forward pass, for each observation in the training set, the values of the input variables in the input layer pass forward through the network, through the hidden layers, and produce predicted output values at the end. All the weights are fixed during the forward pass. That is, at each iteration, the predicted output values are calculated given the input values and the synaptic weights determined by the previous iteration. In the backward pass, the errors between the predicted output value and the corresponding actual output value are calculated. The errors are flowed backward through the network and the weights are adjusted to have smaller errors in the next iteration.

We more formally describe the back-propagation algorithm applied to the multilayer perceptron in Fig. 18.3. The error signal at the output for the nth training observation at iteration t, $e_n(t)(n = 1, \ldots, N)$ is defined by $e_n(t) = d_n(t) - y_n$ where y_n is the derived output value from the neural network for the nth training observation and $d_n(t)$ is the corresponding actual output (or response) value. We now define the total error at iteration t as

$$E(t) = \frac{1}{2}\sum\nolimits_{n=1}^{N} e_n(t)^2 = \frac{1}{2}\sum\nolimits_{n=1}^{N} [d_n(t) - y_n]^2 \tag{18.7}$$

where N is the total number of training example. The scaling factor 1/2 in Equation 18.7 is included to simplify matters in subsequent analysis (Haykin 1999). That is, differentiating Equation 18.7 with respect to $\mathbf{w}$, we have $\partial E(t)/\partial \mathbf{w} = \sum e_n(t)[\partial e_n(t)/\partial \mathbf{w}]$. Note also that the total error E is a function of all the free parameters (i.e., synaptic weights) of the network. For a given training set, E represents the cost function as a measure of learning performance. The objective of the learning process is to adjust the free parameters of the network to minimize E. That is, the synaptic weights are adjusted to make the actual response of the network move closer to the desired response in terms of squared error.

Rumelhart et al. (1986) have employed the generalized delta rule in adjusting weights that is similar to methods of nonlinear optimization employed in statistics. Hence, the goal of the algorithm is to find the optimal weights minimizing the total error E for a given training set. The optimal weights are found in an iterative way since the objective function is highly nonlinear. The generalized delta rule states the following adjustments for the synaptic weights at correction. The correction $\Delta w_{ji}(t)$ to the weight $w_{ji}(t)$ at iteration t is given by

$$\Delta w_{ji}(t) = \alpha \Delta w_{ji}(t-1) - \eta \frac{\partial E(t)}{\partial w_{ji}(t)} \tag{18.8}$$

where α represents the momentum parameter, η is the learning-rate parameter, and $\partial E(t)/\partial w_{ji}(t)$ is the partial derivative of the total error (Equation 18.7) with respect to the weight $w_{ji}(t)$. The generalized delta rule becomes the delta rule when the momentum parameter α is set to zero.

In Equation 18.8, the partial derivative $\partial E(t)/\partial w_{ji}(t) = -\sum_{n=1}^{N} e_n(t)$ $[\partial y_n / \partial w_{ji}(t)]$ represents a sensitivity factor that determines the direction of search in weight space for the synaptic weight $w_{ji}(t)$. The minus sign (before η) ensures that the newly calculated weight will be in the opposite direction from which the partial derivative increases. That is, if the partial derivative is positive, that means that increasing the weight increases error. Hence we will want to change the weight in the negative direction, and the negative sign ensures this. The learning-rate parameter η controls the magnitude of the weight changes from iteration t to iteration $t+1$. A small value of η will lead to a large number of iterations, while too large a value of η will result in a network missing actual minima. The role of the momentum parameter α is to increase the rate of learning but avoid the possible problem of instability (Haykin 1999). Often restricted to [0, 1) in practice, the momentum parameter measures how much the previous weight change influences the current weight change. Hence, a large value for α means the algorithm will tend to maintain the same direction of change as before, i.e., estimation with high momentum means the algorithm will respond slowly to training sample that suggest the reverse of weight change (Berry and Linoff 1997). In other words, assigning a large value for the momentum parameter will have a stabilizing effect in the direction in which weights are change, avoiding oscillations in the estimated parameter.

18.3.3 Application to Credit Scoring

In this section, we apply a multilayer feed-forward network to the credit scoring model. Kindly provided by Professor Fahrmeir, Institute of Statistics, University of Munich, Germany, the data set consists of credit behavior for 1,000 customers of a German bank. The data will be randomly divided into two equally sized groups, 500 customers for the estimation sample and 500

Table 18.1 Estimation results for multilayer perceptron (w_{ji}'s)

	First hidden neuron ($\mathbf{w_{1i}}$)	**Second hidden neuron** ($\boldsymbol{w_{2i}}$)	**Output (DEFAULT)** ($\boldsymbol{w_{3i}}$)
bias	5.65	9.23	
SEX	0.44	−12.78	
MARRIAGE	0.06	−18.39	
BAD	0.39	13.43	
GOOD	−12.48	0.96	
DURATION	−4.03	−2.68	
PAY	−8.02	−19.27	
PRIVATE	0.13	−15.99	
CREDIT	5.28	−16.37	
Bias-h			0.33
h_1			−0.99
h_2			−1.27

customers for the validation sample. The dependent variable (DEFAULT) measures creditworthiness of each customer that is coded 1 if s/he is not creditworthy and 0 if s/he is creditworthy. There are 8 independent variables. Two variables, SEX (female/male) and MARRIAGE (marital status), are customers' demographic characteristics. The rest of the variables represent previous customer behavior and credit characteristics: BAD (bad account), GOOD (good account), DURATION (duration of credit in months), PAY (payment of previous credits), PRIVATE (professional/private use) and CREDIT (line of credit). For more detailed description on variables, see Kim and Shin (1998).

The multilayer perceptron with a hidden layer is applied. The hidden layer has two neurons. In result, our multilayer perceptron is an 8-2-1 network. Using SAS Enterprise Miner, the weights are estimated by the back-propagation algorithm. The estimation results are summarized in Table 18.1.

Neuron 1 in the hidden layer receives 8 input values (plus a bias or an intercept term) from the input layer, and their associated weights are estimated to be the values in the second column in Table 18.1. Similarly, neuron 2 in the hidden layer receives the same input values from the input layer, and the corresponding weights are estimated to be the values in the third column in Table 18.1. Applying the combination and the (logistic) activation function into the input values with the associated weights, we have the output values of neuron 1 (h_1) and 2 (h_2). The resulting output values, h_1 and h_2, become the input values to the output layer. That is, an output neuron (DEFAULT) in the output layer receives two input values (plus an intercept term) in the hidden layer, and the corresponding weights are estimated to be the values in the fourth column in Table 18.1. As you can see, it is very difficult to interpret the weights directly. For example, the w's for few of the variables have opposite signs in the two neurons.

18.3.4 Optimal Number of Units in the Hidden Layer, Learning-Rate, and Momentum Parameters

ANNs are often criticized for their subjectivity in specifying network architecture and training parameters (Tam and Kiang 1992). Model selection has been the one of the most difficult problems in statistics. Similarly, selecting the best network architecture in ANN is not a trivial problem. The universal approximation theorem suggests that the multilayer perceptron with one hidden layer will perform satisfactorily. Still we need to determine the number of neurons in the hidden layer. Large numbers of hidden layers will be able to capture the sophisticated relationship between input and outputs. However, too many hidden layers will essentially memorize the training observations, leading to overfitting.

There are heuristic methods for determining the number of neurons in the hidden layer. In order to avoid over-fitting, they should not be more than twice the number of input variables in the input layer (Berry and Linoff 1997). Others use the square root of the number of input variables in the input layer as the number of neurons in the hidden layer (Kim et al. 2005).

The better way of finding the optimal number of units in the hidden layer is to find it empirically. We partition the data into the estimation and the validation samples. Then we train networks with different number of neurons in the hidden layer using the estimation sample and evaluate the trained networks in the validation sample. The network with the smallest SSE (or the highest hit-rate for the classification task, see Chapter 11) is chosen as the best network architecture that has the optimal number of units in the hidden layer.

We can also conduct a grid search to determine the optimal number of neurons in the hidden layer simultaneously with the optimal learning-rate and momentum parameters. For example, we may try out $\eta \in \{0.1, 0.4, 0.7, 0.9\}, \alpha \in \{0.0, 0.2, 0.6, 0.9\}$ and the number of neurons in the hidden layer $\in \{1, 2, 3, 4, 5\}$. Then you need to train $80 (= 4 \times 4 \times 5)$ different networks and compare their prediction performances.

18.3.5 Stopping Criteria

Similar to all nonlinear optimization problems, there are no clean-cut criteria for stopping the weight adjustments in the back-propagation algorithm. However, there are a couple of reasonable criteria. First, the objective function of the multilayer perceptron, Equation 18.7, will have a local or global minimum when the first-order partial derivatives $\partial E/\partial w_{ji}$ equal to zero for all i and j. Therefore, we can formulate a stopping rule with respect to the gradient vector of weights (Haykin 1999). We stop the iteration when the Euclidean norm of the gradient vector reaches a small gradient threshold specified by the user.

The limitations of the gradient method for the stopping rule are the length of running time with the requirement of computing the gradient vector. Alternatively, we can propose a stopping rule based on the rate of change in the objective function itself because it is stationary at the minimum (Haykin 1999). We stop the training iteration when the absolute rate of change in the objective function is sufficiently small.

Both these rules require subjectively determined thresholds, and undoubtedly the appropriate thresholds depend on the type of data being modeled. Researchers should try alternative values and settle on "default values" as they gain experience with their data.

18.3.6 Feature (Input Variable) Selection

Selecting relevant input variables is important in improving the performance of a neural network. The objective is to find the minimum subset of input variables that yield the highest accuracy. This problem, often called feature subset selection, is conceptually similar to the variable selection problem in a classical regression model (see Chapter 11). Interest in the feature selection problem is intensifying because the size of customer information file is increasing.

There are two types of approaches to finding an optimal feature subset in a neural network. The "filter" approach performs feature selection independently of the neural net learning algorithm. In contrast, the "wrapper" approach finds the optimal subset of features guided by the performance of the learning algorithm. The filter approach is computationally more efficient than the wrapper approach. However, many researchers have criticized the filtering model since it ignores the effect of the selected features on the performance of the neural network (Yang and Honavar 1998; Hsu et al. 2002). The wrapper model overcomes this problem of the filtering approach, but can be computationally expensive since each candidate feature subset must be evaluated in estimating the neural network.

Several algorithms have been proposed to speed up the computation in the wrapper approach. For example, Richeldi and Lanzi (1996) partitioned features into a number of groups (called factors) and employed a genetic algorithm to explore the feature space originated by the factors and determine the set of the most informative feature configurations. On the other hand, Setiono and Liu (1997) proposed a method that added a penalty term to the error function of the neural network and identified redundant network connections from those relevant ones by their small weights when the network training process was completed.

We briefly describe the wrapper approach proposed by Hsu et al. (2002). Their feature selection model, called the "artificial neural net input gain measurement approximation" (ANNIGMA), performed better than two

benchmark wrapper models. They successfully applied their model to two real-world dataset, the one with 192 features and the other with 41 features. The ANNIGMA wrapper's method of finding an optimal feature subset is similar to that of a stepwise variable selection in a classical regression. It starts with a complete set of original features (or input variables) and removes features from candidate subsets during search. They presented three versions of their algorithm: (1) greedy backward elimination (BE), (2) backward elimination with backtracking (BEB), and (3) backward stepwise elimination (BSE). BE starts with estimating a neural net model of all features and obtains ANNIGMA scores of each feature measuring the relevance (or importance) of a feature to the performance of the neural net model. It repeatedly eliminates the next worst ANNIGMA ranked feature until the error rate of the neural net goes up. BEB allows for backtracking. That is, if the error rate goes up, the previous feature eliminated is restored and the next worst ranked feature is eliminated. The process is iterated until a performance-improving elimination is found for each size of feature subsets. Finally, BSE is designed to speed up feature selection for large databases. It eliminates a large number of seemingly irrelevant features in early cycles (i.e., employing BE) and adjusts the feature subset carefully in the subsequent cycles (i.e., employing BEB).

18.3.7 Assessing the Importance of the Input Variables

As we saw in Sect. 18.3.3, it is virtually impossible to interpret directly from the estimated w's how important each input variable is in its impact on the output variable. This is because the inputs influence several hidden neurons, and the signs can be in opposite directions. Even if the signs are the same, it is still difficult to compare one input variable to another based on the several estimated w's. For example, one variable may have a high w linking to one neuron, but a low value linking to another. Is this variable more important than one that has relatively moderate w's linking to each neuron?

There is no easy solution to this problem. A common approach to assessing variable importance is some form of sensitivity analysis (Berry and Linoff 1997). For example, to assess the importance of input variable X_1, one might fix the other variables at their means and then vary X_1 over its relevant range. One could graph the dependent variable or calculate a statistic to show how much the dependent variable changes. The problem with this approach is that nonlinearity of the neural net captures many interactions between the inputs, i.e., the impact of X_1 when the other variables are at their means may be very different than if the variables are at some different value.

Another approach is simply to graph the dependent variable as a function of the input, without doing any calculations from the neural net model. This

can provide insight similar to a correlation, but as with sensitivity analysis, does not show the rich non-linearity and interactions that the neural net has derived. It also does not control, in the regression sense, for the other variables in the model.

Despite these difficulties, we recommend that the user examine at least some measure of relationship between the inputs and the output, whether it is calculated via sensitivity analysis or simply by graphing the independents against the dependent. This at least provides some idea as to the nature of the relationship that is in the data.

18.4 Radial-Basis Function Networks

The discipline of neural networks is broad, covering diverse classes of models. ANNs can be grouped into models for supervised learning (also called learning with a teacher) and for unsupervised learning (also called learning without a teacher). The most popular neural network models for the supervised learning tasks are the multilayer perceptron and a radial-basis function network while self-organizing map (SOM) may be the most well known neural net model for the unsupervised learning tasks. We cover the SOM in the Clustering chapter.

18.4.1 Background

In this section, we introduce one more neural net model for supervised learning called a radial-basis function (RBF) network that is recently attracting more attention (Poggio and Girosi 1990; Park and Sandberg 1991; Abdi 1994). After multilayer perceptrons, an RBF network is the most popular neural network model for supervised learning. Different from a multilayer perceptron, a radial-basis function network fundamentally views the design of a neural network as a curve-fitting approximation problem in multidimensional space. Its goal is to find the best multidimensional curve explaining the nonlinear relationship between inputs and outputs in the training data. A radial-basis function network can be said to have a more formal mathematical basis for the formulation of the network. As its name implies, a radial-basis function network adopts the theory on radial-basis functions in developing the foundation for the hidden layer. Originally introduced to solve the multivariate interpolation problem, radial-basis functions have become one of the main fields of study in numerical analysis (Light 1992).

A radial-basis function network has a similar architectural design as a multilayer perceptron. It consists of an input layer, a hidden layer and an output layer (see Fig. 18.3). The functions of the input layer are no different from those in a multilayer perceptron. However, two networks differ from

each other in several important aspects (Haykin 1999). First, the activation function of each hidden unit in a radial-basis function network computes the Euclidean distance between the input vector and the center of that unit. In contrast, the activation function of each hidden unit in a multilayer perceptron computes the inner product of the input vector and the synaptic weight vector of that unit. Second, a radial-basis function network is allowed to have only one hidden layer while a multilayer perceptron can have one or more hidden layers. Finally, the hidden layer of a radial-basis function network is nonlinear while the output layer is only allowed to take the linear activation function. As explained later, the role of the hidden layer is quite different from the output layer in a radial-basis function network. However, the roles of the hidden and output layer in a multilayer perceptron are similar, and they are usually nonlinear.

Because of these differences, a radial-basis function network has the advantage of avoiding finding a set of parameters that only represents a local minimum. Applications of a multilayer perceptron often end up in a local minimum and their speed of convergence is sometimes problematic. In a radial-basis function network, the only parameters that are adjusted in the learning process are the linear mapping from the hidden layer to the output layer. Because of the linear activation function, the error surface of the radial-basis function network is quadratic, and hence has a single minimum.

18.4.2 A Curve-Fitting (Approximation) Problem

To fix ideas of radial-basis function network, let us consider a nonlinear regression problem where the value of the dependent variable for observation i ($i = 1, \ldots, N$) is d_i and the corresponding vector of the independent variables is $\mathbf{x}_i$. Then the general nonlinear relationship between the dependent variable and the vector of independent variables can be written as

$$d_i = f(\mathbf{x}_i) + \varepsilon_i \tag{18.9}$$

where ε_i is the random error and $f(\mathbf{x}_i)$ is a smooth curve.

A radial-basis function network approximates the high-dimensional curve $f(\mathbf{x}_i)$ in the Equation 18.9 by $F(\mathbf{x})$.

$$F(\mathbf{x}) = \sum_{j=1}^{m} w_j \varphi_j(\mathbf{x}) \tag{18.10}$$

where m is the number of neurons in the hidden layer and w_j represents the weight from the jth neuron of the hidden layer to the output. The number of neurons in the hidden layer or the number of basis function φ is generally less than the number of data points, $m < N$.

The mathematical justification of a nonlinear transformation followed by a linear transformation in the Equation 18.10 can be found in a number of studies (Cover 1965; Poggio and Girosi 1990). In general, a set of basis functions $\{\varphi_j(\mathbf{x}), i = 1, \ldots, m\}$ is assumed to be linearly independent, and each $\varphi_j(\mathbf{x})$ takes the form of a radial-basis function. That is,

$$\varphi_j(\mathbf{x}) = \varphi(\|\mathbf{x} - \mathbf{t}_j\|) \tag{18.11}$$

where $\|\cdot\|$ denotes an Euclidean norm and $\mathbf{t}_j$ is the center of the radial-basis function for neuron j. One of the most popular choices for the φ function is the Gaussian function.

$$\varphi_j(\mathbf{x}) = \exp\left[\frac{1}{2\sigma_j^2}\|\mathbf{x} - \mathbf{t}_j\|^2\right] \tag{18.12}$$

where σ_j^2 is the variance of the Gaussian distribution. The condition $\sigma_j^2 = \sigma^2$ for all j is often imposed for mathematical simplification. Different learning strategies can be adopted depending on how the centers of the radial-basis functions ($\mathbf{t}_j$) are specified (Haykin 1999). The simplest approach is to choose the (fixed) locations of the centers randomly from the training dataset. A more sophisticated method is to utilize the k-means clustering algorithm which places the centers of the radial-basis functions in only those regions of the input space where significant data are present. Alternatively, the centers of the radial-basis functions (along with other parameters) can be treated as parameters to be estimated.

The role of the activation function of each hidden unit in a radial-basis function network ($\varphi_j(\mathbf{x})$) is somewhat different from its role in a multilayer perceptron. The activation function in a radial-basis function network computes the distance from the input to each of the centers. Each cell of the hidden layer represents a center. On the other hand, the activation function of each hidden unit in a multilayer perceptron computes the inner product of the input vector and the synaptic weight vector of that unit.

Given the above specification, the estimation problem of the radial-basis function network is to determine a set of weights $\{w_j | j = 1, \ldots, m\}$ to minimize the following error function (Haykin 1999).

$$\mathrm{E} = \sum_{i=1}^{N} (d_i - F(\mathbf{x}_i))^2 + \lambda \|\mathbf{D}F(\mathbf{x})\|^2 \tag{18.13}$$

where $\mathbf{D}$ is a stabilizer and λ is the regularization parameter.

As seen in Equation 18.13, the error objective function for a radial-basis function network is different from that for a multilayer perceptron (e.g., backpropagation). The theory behind the derivation of Equation 18.13 is regularization theory by Tikhonov (1963). In order to solve an ill-posed hyper-surface reconstruction problem, he proposed to stabilize (or smooth) the solution by

Table 18.2 Estimation results for radial-basis function network

	First hidden neuron	Second hidden neuron
SEX	0.16	−1.44
MARRIAGE	−1.61	−0.64
BAD	0.80	1.58
GOOD	0.70	0.48
DURATION	−0.16	−0.30
PAY	0.11	−0.28
PRIVATE	0.23	0.88
CREDIT	0.78	−0.06

means of an auxiliary nonnegative functional that embeds prior information about the solution. That is, Equation 18.13, which is called the Tikhonov functional consists of two terms, the standard error term and the regularizing term. The standard error term or $\sum (d_i - F(\mathbf{x}_i))^2$ measures the error (or distance) between the actual response d_i and the estimated response $F(\mathbf{x}_i)$. The regularizing term or $\lambda||\mathbf{D}F(\mathbf{x})||^2$ represents a model complexity-penalty function. The regularization parameter λ controls the balance between the importance of the training examples and the prior smoothness constraint. If λ is close to zero, the solution is mostly determined from training examples (i.e., less smoothing). As λ gets larger, training examples are more or less treated as unreliable. In addition, the stabilizer $\mathbf{D}$ represents prior information about the form of the solution.

18.4.3 Application Example

We apply a radial-basis function network to the same credit scoring data used in the example for the multilayer perceptron. We assume a hidden layer with two neurons and a Gaussian radial-basis function given by Equation 18.12. We also assume that $\sigma_j^2 = \sigma^2$. Using SAS Enterprise Miner, the center of the radial-basis function ($\mathbf{t}_j$) is determined and the results are summarized in Table 18.2.

The numbers in the second column of Table 18.2 are the centers of the Gaussian radial-basis function for the first hidden neuron. Similarly, the numbers in the third column are the centers of the Gaussian radial-basis function for the second hidden neuron. Similar to the results from multilayer perceptron, the training or estimation results are difficult to interpret.

We also applied the radial-basis function networks (along with multilayer perceptron and a logit model) to the validation sample of 500 customers. The hit ratio of the RBF networks is 79.2% (396/500) whereas the hit ratios of multilayer perceptron and a logit model are 79.6% (398/500) and 76.4% (= 382/500) respectively. We conclude that neural network models show

slightly better prediction performances than a logit model. But there is no difference between the RBF networks and multilayer perceptron. There have been few researches that directly compared the RBF networks with multilayer perceptron. For example, Park et al. (2002) found that the RBF network is simpler to implement than multilayer perceptron, needs less computational memory, converges faster, provide slightly better predictions and global minimum convergence is achieved.