Analytical Cellular Pathology 14 (1997) 19–30
IOS Press

Opening the black box: the relationship between neural networks and linear discriminant functions

Roger A. Kemp*, Calum MacAulay and Branko Palcic
BC Cancer Research Centre, Vancouver BC, V5Z1L3, Canada

Received 29 November 1996
Revised 14 February 1997
Accepted 19 March 1997

Abstract. Over the last ten years feed-forward neural networks have become a popular tool for statistical decision making. During this time, they have been applied in many fields, including cytological classification. Neural networks are often treated as a black box, whose inner workings are concealed from the researcher. This is unfortunate, since the inner workings of a neural network can be understood in a manner similar to that of a linear discriminant function, which is the standard tool that researchers use for decision making.

This paper discusses feed-forward neural networks and some methods to improve their performance for classification problems. Their relationship to discriminant functions will be examined for a simple two-dimensional classification problem.

Keywords: Neural network, linear discriminant function, classification, nonlinear classifier

1. Introduction

Neural networks and other connectionist models have recently received much attention as a preferred solution to nearly every classification problem. The field has expanded rapidly, with neural networks being touted for classification, time-series prediction, process control and other applications. This has led to a misperception among many that neural networks are the panacea to decision making problems. This is due, in part, to the neural network terminology, which is drawn from biological terms applied to the brain. This terminology gives the connotation that neural network models are a software version of the brain, inflating our expectations.

Pattern classification is a task performed by the brain all the time. Images are recognized by the human brain in a fraction of a second. Yet, the firing rate of neurons in the brain is of the order of a few milliseconds. This means that the brain can recognize an image in less than 100 neuron cycles. Switching times in today's computers operate roughly a million times faster than the brain. Yet the most sophisticated pattern recognition software running on the fastest computers cannot come close to what the brain does routinely. It is the distributed nature of the brain that allows it to process

*Corresponding author: Roger A. Kemp, Cancer Imaging, BC Cancer Research Centre, West 10th Avenue 601, Vancouver BC, V5Z1L3, Canada. Tel.: +1 604 877 6010; Fax: +1 604 875 6857.

images quickly and classify them. This distributed processing feature is the motivation behind using neural network models for classification. The brain contains approximately 10^{11} neurons each with the order of 10^4 interconnections. Each neuron performs a nonlinear transformation on a weighted combination of the electrical signals it receives from other cells. The term neural networks has grown to describe a variety of systems that have this property: i.e., interconnected elements each performing simple computations on their inputs.

It is important to keep in mind that the main similarity between the neural networks implemented on computers and those of the brain is the dense interconnectedness of the elements. In other respects, artificial neural networks bear little resemblance to neural networks of the brain. The brain's neurons do not simply return an output voltage, but instead produce a pulse or series of pulses in response to a stimulus. Further, the propagation of an error signal, used by most neural network training algorithms to update the network weights, is nothing like the manner in which the neural networks of the brain learn.

These differences are not an impediment to using neural networks as a classification tool. The reason why we use neural network algorithms is because they are flexible algorithms that allow us to encode nonlinear relationships between input and response variables.

2. Feed-forward neural networks

One of the most popular neural network schemes is the *feed-forward neural network* trained with *back-propagation* of the training error [13]. This type of network is also known as a *multi-layer perceptron* after terminology introduced by Rosenblatt [12]. The structure of such a network is shown in Fig. 1. This network consists of inputs, an intermediate layer of processing elements called hidden units and outputs.

The hidden units give the neural network the ability to encode nonlinear relationships between inputs and outputs. As will be explained, they receive a score from one weight layer and pass it to the next weight layer. Since they represent an intermediate step in the whole calculation, their output values are not directly observed by the user; hence their name. In Fig. 1, there are two weight layers connecting the processing elements. This is called a two-layer network or a network with a single layer of hidden units.

In Fig. 1 the inputs are shown by x_i, the hidden units by V_j and the outputs by O_k. The inputs are connected to the hidden units by connection weights w_{ij}, and the hidden units are connected to the outputs by connection weights W_{jk}. For a given input pattern $\mathbf{x}^\mu$, the output value of the hidden unit is given by

$$V_j^\mu = g(h_j^\mu) = g\left(\sum_i w_{ij} x_i^\mu\right), \tag{1}$$

where $g()$ is a nonlinear function. Thus, each hidden unit receives a weighted sum (h_j^μ) of the inputs and passes this through the nonlinear function $g()$. The resulting value (V_j) is made available to the next layer of weights.

In the same manner, the outputs O_k are given by

$$O_k^\mu = g\left(\sum_j W_{jk} V_j^\mu\right) = g\left(\sum_j W_{jk}\, g\left(\sum_i w_{ij} x_i^\mu\right)\right). \tag{2}$$

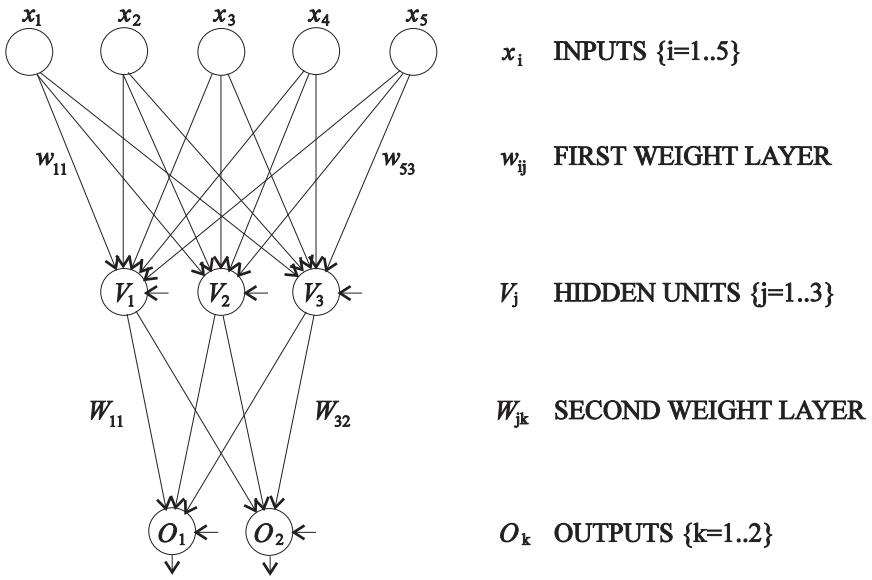

Fig. 1. Structure of a feed-forward neural network with one layer of hidden units. Biases to non-input units are shown as horizontal arrows leading into the units.

This process is repeated for as many layers of weights as exist in the network.

The output values from the final layer of the neural network will be continuous. When using the neural network as a classifier, one specifies a rule for interpreting the output as a class membership. In the case where a single output is used, the output value is compared to a threshold and assorted into the appropriate class.

The function $g()$ is referred to as the activation function. The most commonly used activation function is sigmoidal in shape [6],

$$g(h) = \frac{1}{1 + \mathrm{e}^{-h}}, \tag{3}$$

and is shown in Fig. 2. It saturates for small or large inputs. This means that when h has a large magnitude (positive or negative), the function $g(h)$ returns a value that asymptotically approaches its minimum or maximum value. The nonlinear nature of the activation function is the key feature which gives feed-forward neural networks the ability to perform mappings that cannot be made with a discriminant function.

It is usual for each non-input unit in the network to have a bias or threshold term. Standard practice is to add an artificial input unit which has a constant value 1 which is connected to each non-input unit. These connection weights are treated in the same fashion as all other weights when updating the values of the weights. By implementing the biases in this manner, the summations in Eqs (1) and (2) can include the extra weights (which are multiplied by 1) without changing the form of the equation. In Fig. 1 the biases are shown as horizontal arrows into the non-input units.

The process of training the network starts with a set of training examples $(\mathbf{x}^\mu, \mathbf{y}^\mu)$, where $\mathbf{x}^\mu$ are the training pattern inputs and $\mathbf{y}^\mu$ are the desired outputs. For a typical two class decision task, a single

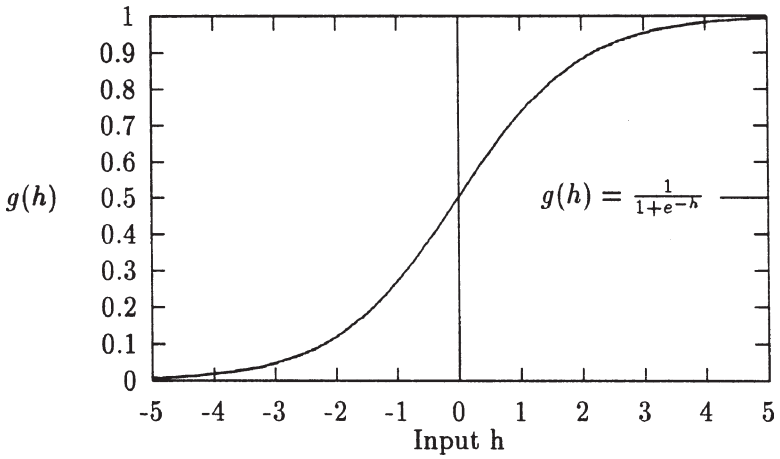

Fig. 2. A sigmoidal shaped function (one that saturates for extreme input values) is the most common nonlinear activation function used with neural networks.

output unit is all that is needed. Since this output unit returns a value between 0 and 1, its value can be used as a measure of the correspondence of the example to a class. When using a neural network to separate objects into more than two classes, it is standard to use as many output units as there are classes. For a training example from the nth class, one then trains the network to output 1 for the nth output unit and 0 for all others.

For the binary decision task, the single output y^μ for each training pattern is set to either 0 or 1, according to the known class of the example. The neural network weights are initialized to small random values. The training examples are then presented to the neural network inputs and the outputs are calculated from Eq. (2). The performance of the network on the training set is assessed using an error function. The standard error function is the sum of squares measure

$$E(\mathbf{w}) = \frac{1}{2}\sum_{\mu k}\left(y_k^\mu - O_k^\mu\right)^2. \tag{4}$$

It is possible to calculate the derivative of the error function with respect to the training weights, **w**, using the chain rule from calculus [6]. This derivative is then used by the training algorithm to adjust the weights to make a slight reduction in the overall classification error. Therefore, the problem of creating a feed-forward neural network with good performance essentially becomes a nonlinear regression problem. The back-propagation algorithm is simply the process of calculating the error gradient and taking small steps along the gradient. This is an inefficient way to solve a nonlinear optimization problem. It may require thousands of iterations of the algorithm to find a network that gives satisfactory classification rate for the training set.

Many authors have proposed improvements to back-propagation, such as momentum terms and adaptive step sizes [7]. An alternative is to use an minimization algorithm that uses an approximation to the second derivative of the error function. This can be accomplished using a quasi-Newton algorithm or conjugate gradient algorithm such as the Polak–Ribiere method [10]. Such methods have been found to be an order of magnitude more efficient than standard back-propagation for training neural networks [8,14].

3. Decision surfaces

Figure 3 shows three different neural network structures, each with two inputs and a single output. For the purpose of illustrating the decision regions created by classifiers, all the classifier examples in this paper have two inputs. Features 1 and 2 are presented as inputs to the classifiers generating a single real-valued output. In two dimensions, decision surfaces will always be some kind of curved line separating the classes.

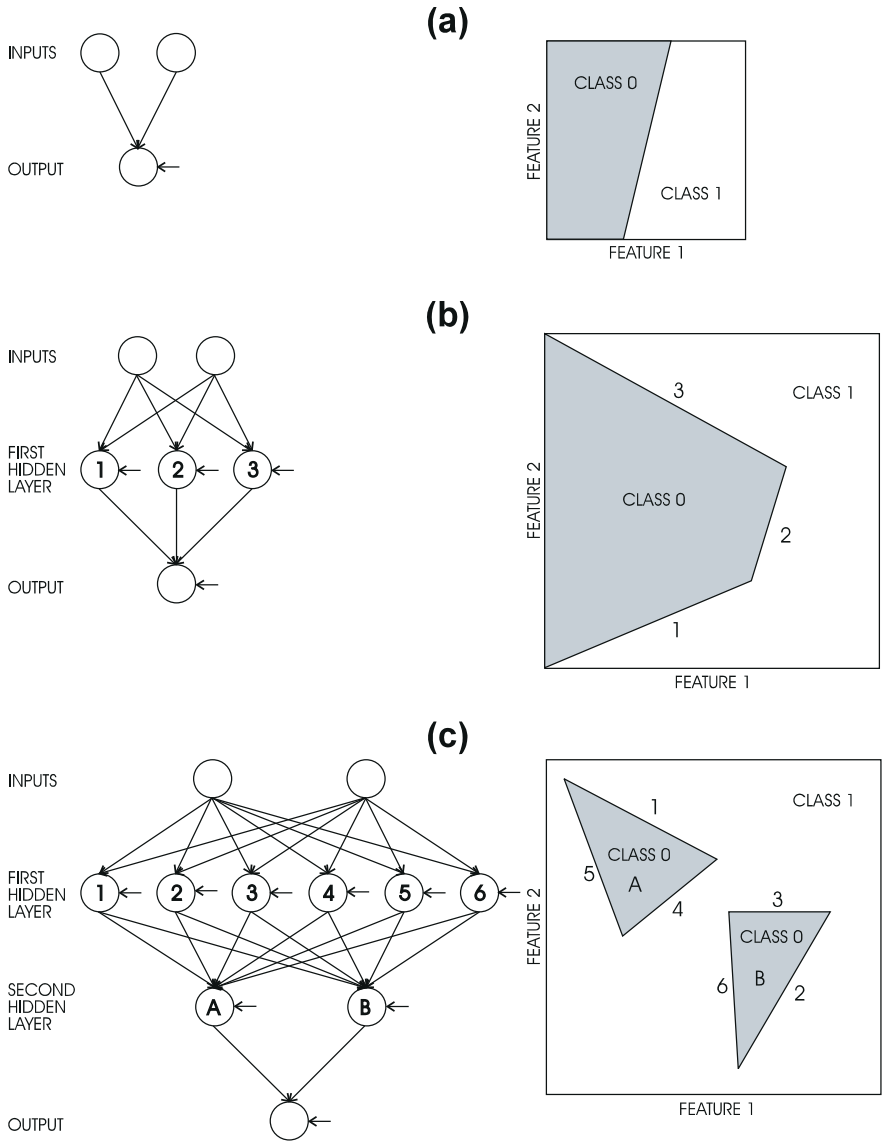

Fig. 3. Possible decision surfaces are shown for feed-forward networks with one, two and three layers of weights. In the first case (a) the decision surface will be a line and the network performs much like a linear discriminant function. When there are two layers of weights (b) the decision surface will be curved and can separate a single region of arbitrary shape. If more than two weight layers are used (c) the network can form boundaries around multimodal distributions in the feature space.

For the neural networks shown in Fig. 3, the number of hidden unit layers, and thereby the complexity, of the network is different in each case. If we plot the output of the networks over a range of input values and connect the points whose output is equal to the decision threshold, we obtain the decision boundary for the network. Possible mappings for the three neural networks in the figure are shown on the right. The shaded areas correspond to regions of the feature space for which the neural network output is less than the decision threshold (usually 0.5).

The simplest neural network structure possible is one with no hidden units, as shown by Fig. 3a. In this case the output is simply a weighted sum of the inputs. Since the sum of squares error measure is normally used (4), training this neural network is equivalent to performing linear regression on the training data. The decision surface for this network can be visualized as a line in the two-dimensional feature space. The line is determined by the vector containing the two network weights as components, and the intercept is given by the node bias (shown in the figure as a horizontal arrow entering the output unit).

Under certain conditions the linear classifier generated by this neural network is identical to Fisher's linear discriminant function (LDF). This is the case when the training examples from the two classes are normally distributed with identical covariances and equal prior probabilities. Further, an equal number of training examples must be used from each class to train the classifier [11].

If we form a linear combination of the transformed output of three such linear classifiers and apply the activation function (2), we obtain the network structure shown in Fig. 3b. This neural network contains a single layer of three hidden units. Each unit roughly corresponds to a segment of the nonlinear boundary between the two classes. The corners of the boundary will be rounded because contributions from pairs of hidden units to the output will shift the position of the threshold [9]. If the weights connecting inputs to hidden units are scaled up by some large value, the decision boundary will approach arbitrarily close to the boundary in the figure. It has been shown that a neural network with a single layer of hidden units can represent any continuous function to an arbitrary accuracy given enough hidden units [4].

When two layers of hidden units are used it is possible to approximate multimodal decision surfaces (Fig. 3c). The neural network shown contains two layers of hidden units with six units in the first hidden layer and two in the second. Points in the shaded regions A and B are mapped by the neural network into class 0 while those outside belong to class 1. The figure on the right-hand side shows a possible decision surface for such a neural network structure. Such a mapping would arise if the weight values connecting units 1, 4 and 5 to unit A were comparatively large, while those connecting 2, 3 and 6 to A were small. In a similar manner, the weights between units 2, 3, 6 and unit B would also have a large magnitude while the weights between units 1, 4, 5 and unit A would be small.

Figure 3c shows the class 0 decision regions as triangular regions with sharp corners. There is nothing special about triangles; the shape of the decision boundaries depends completely on the values of the weights in the network. Further, the boundaries will tend to be smooth curves, especially at the start of the training procedure. Sharp corners will only occur if the network weights are optimized for many iterations, long enough so that the training algorithm attempts to fit individual examples.

4. A two-dimensional classification example

Consider the simple two-dimensional classification problem shown in Fig. 4. The figure shows 200 data points from two annular uniform probability distributions for objects labelled class 0 and class 1. The densities are selected to overlap slightly to give rise to the possibility of selecting objects with

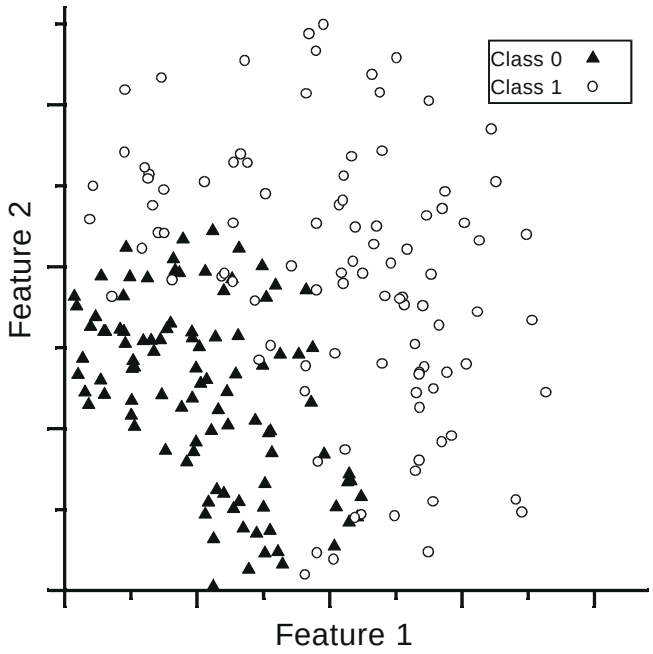

Fig. 4. Training data for a simple classification problem. The points are randomly selected from two slightly overlapping annular probability distributions.

nearly identical features from different classes. We randomly generate 200 points, 100 from each class, as the training data. A test set of 1000 points is generated to test the performance of classifiers designed with the training data. It consists of 500 points from each class drawn from the respective probability densities.

The linear discriminant boundary for this problem is shown in Fig. 5. It correctly separates 81.5% of the 200 training examples and 83.7% of the test examples. We can improve upon this classification rate by calculating a quadratic discriminant function. This function contains squares of the feature variables as well as the crossterm of the inputs. For this example it does a better job of approximating the curvature of the decision boundary between the two classes. It correctly separates 85.5% of the training examples and 86.9% of the test examples.

However, the curse of dimensionality can make it difficult to use quadratic discriminant functions. For a classification problem with d features, the quadratic discriminant function has $\frac{1}{2}(d+1)(d+2)$ free parameters. For a small feature set, say 10 features, the quadratic discriminant function will have 66 parameters – a manageable number. For a feature set with 50 features, however, the number of free parameters in the discriminant function jumps to 1328, making it more difficult to design a robust classifier. Even if one uses a stepwise selection procedure, with so many features to choose from, it is likely that a quadratic feature which proves useful for separating the training data will not perform well on a test set.

Figure 6 shows the decision boundary for this example generated by a neural network with two hidden units. The neural network was created by starting with a network of random weights and optimizing them with the Polak–Ribière conjugate gradient optimization routine [10]. The training set was presented to the network 200 times (i.e., 200 iterations of the algorithm). The optimized network correctly classifies 89.0% of the training examples and 88.1% of the test set.

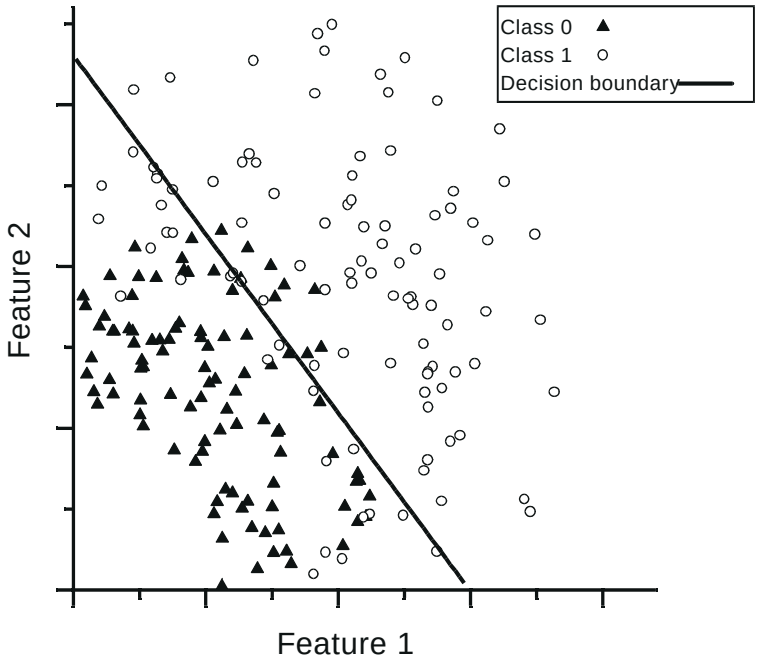

Fig. 5. The decision boundary is shown for the linear discriminant function.

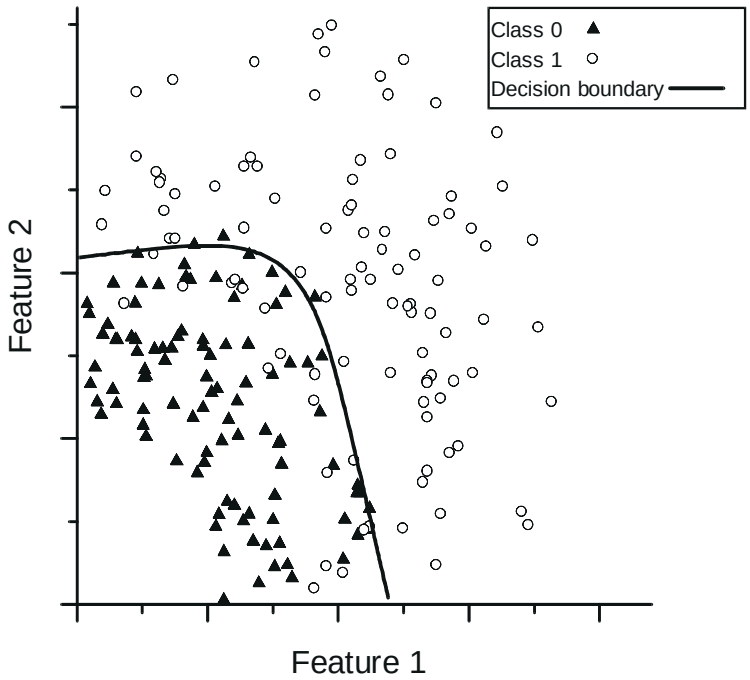

Fig. 6. The decision boundary is shown for a neural network classifier with two hidden units.

Figure 7b shows the structure of a network with two hidden units that was applied to the classification problem. We can treat the pairs of weights connecting the inputs to each hidden unit as a two-component vector. If we plot the vector of weights corresponding to each hidden unit, we obtain the pair of vectors shown in Fig. 7a. This figure shows how the decision boundary is created as the result of the influences of each of the hidden units acting much like a discriminant function. Hidden unit 1 describes the upper left portion of the decision boundary and is dominant in classifying training examples in this area. The dashed line (perpendicular to the vector) is defined by the vector (w_{A1}, w_{B1}) along with bias 1. The bias acts in a similar manner to the constant in a linear classifier, merely shifting the position of the line from the origin. Hidden unit 2 describes the lower right portion of the decision boundary.

There are two ways to improve the classification rate of the neural network solution to this problem. The first is to repeat the training process with more iterations. The performance of the previous solution, obtained after 200 iterations of a conjugate gradient algorithm, could be improved slightly by optimizing it further. In this case the improvement is marginal, and there is no visible difference between the decision boundaries created by the two neural networks.

A second way is to use a larger number of hidden units. Adding extra hidden units increases the number of parameters in the problem, thereby increasing the network's ability to fit the training

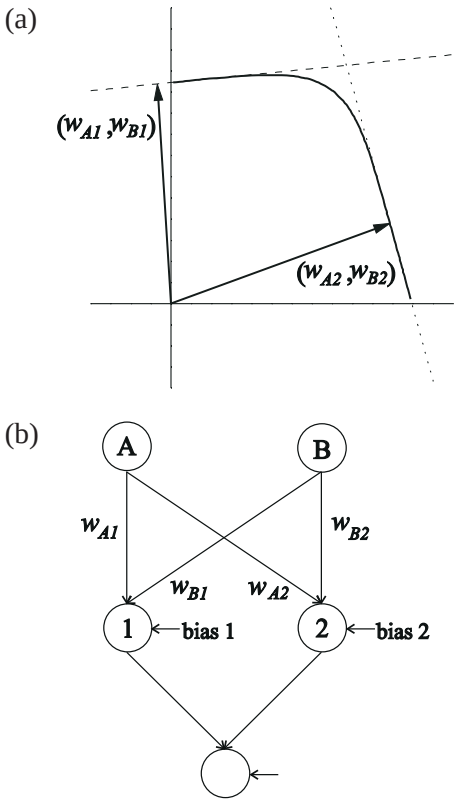

Fig. 7. The decision boundary is shown (a), for a network with two hidden units (b). If we plot the weights leading into the hidden units as vectors in the feature space, we obtain the vectors shown. This shows how each hidden unit behaves as a discriminant function for objects in one area of the feature space. The dotted lines are the discriminant functions determined by the respective vectors and biases shown in (b).

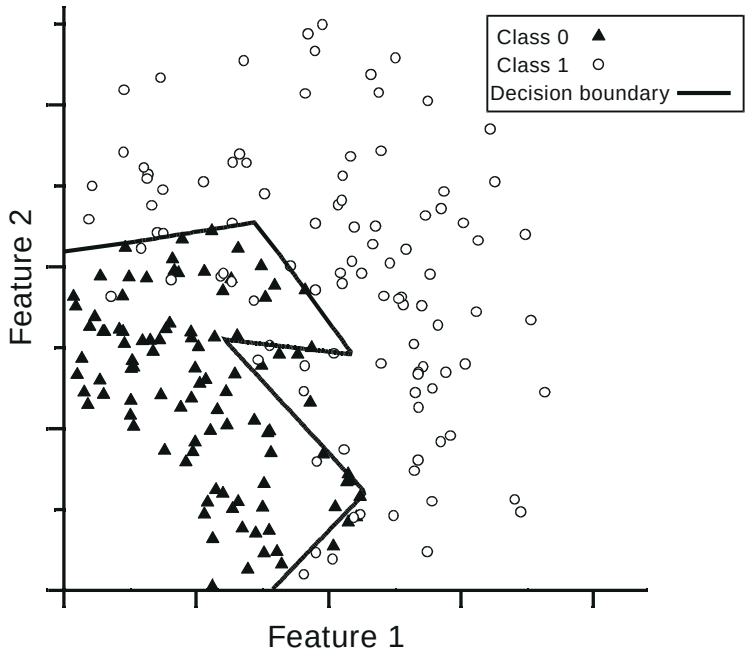

Fig. 8. The decision boundary is shown for a neural network classifier with five hidden units.

Table 1
The classification rates of the discriminant functions and neural networks (NN) is shown for the two-dimensional classification problem

Classifier	Training set performance	Test set performance
Linear discriminant	81.5%	83.7%
Quadratic discriminant	85.5%	86.9%
NN with 2 hidden units	89.0%	88.1%
NN with 5 hidden units	92.5%	86.0%

examples. Figure 8 shows the decision boundary for a neural network with five hidden units. It was obtained by optimizing a randomly initialized network with 10,000 iterations of the conjugate gradient algorithm described earlier. Due to the large number of presentations of the training patterns, the training procedure produced a network that is precisely tuned to the training set. The decision boundary appears to be a series of five segments, each due to the influence of one of the hidden units. There is almost no rounding of the corners where the portions of the decision boundary corresponding to each hidden unit meet. This occurs due to the large network weights that can result from training the network for many iterations. Intuitively, this decision boundary is actually worse given what we know of the original data distribution.

The network with five hidden units correctly classifies 92.5% of the training examples, but only 86.0% of the test examples. This should be compared with the classification rates of the earlier classifiers, all of which are shown in Table 1.

Thus the classification rate of the five hidden unit network is worse than that of the two hidden unit neural network. This discrepancy is an example of what is referred to as *overfitting* the data. The large number of parameters in the model allows the network to give undue attention to individual training examples. Its training set classification rate is better than that of the simpler neural network,

but no longer reflects how well it will perform on a separate test set. The five hidden unit network contains 21 weights. Since there are 200 training examples, the ratio of training examples to free parameters is nearly 10 : 1, yet the network appears to overfit the data.

5. How many hidden units should one use?

The most commonly asked question when neural networks are used is how many hidden units are required. In light of the discussion above, we see that it is not possible to formulate such rules based strictly on the number of inputs/outputs and training examples. The optimal number of hidden units is determined by the relationship between the groups being classified. In the simplest case, the two groups are linearly separated in the feature space and no hidden units are required, no matter how many training examples are used.

When too many hidden units are used, the training set classification rate tends to be higher than the rate found by any validation procedure. The difference between these rates is termed shrinkage [2]. It is possible to reduce the shrinkage due to overfitting by using a large training set. Baum and Haussler derive a lower bound on the size of a training set to minimize the error on a test set [3]. They have shown that if one trains a network of W weights and desires a test set error rate of less than ε, one needs of the order of W/ε training examples. This has led to a rule of thumb that at least ten training examples are required for each weight in the network [2].

It is possible to obtain a measure of the optimal number of hidden units from an information theoretic point of view. Fogel derives an information statistic that provides an objective measure of the suitability of a particular network for a binary classification problem [5]. This statistic, derived from Akaike's information criterion, represents the mean log-likelihood of a statistical model [1]. For a given neural network and data set, Fogel's information statistic returns a single likelihood value [5]. One trains networks with different architectures on the same training set and computes the likelihood values for the different networks. By comparing the relative likelihoods it is possible to select the best network to solve a classification problem.

6. Conclusions

There is a lot of hype in the neural network field, and this has fueled unrealistic expectations of their usefulness in statistical decision making. The feed-forward neural network, the most commonly used neural network paradigm for classification problems, is a flexible, nonlinear regression model. It can be used to represent aribitrarily complicated decision boundaries between data distributions and is subject to the same distributional assumptions as standard statistical methods. The process of training neural networks, whether by simple backpropagation or by some more efficient algorithm, amounts to performing regression on a nonlinear function of connection weights.

The form of a feed-forward network can be viewed as an extension of a discriminant function. As Fig. 3b shows, each hidden unit gives rise to a portion of the decision boundary between the data classes. As one adds more hidden units to a network the ability of the network to fit the data increases. With enough hidden units it is possible to obtain a perfect classification of any training data. For practical problems, however, there is always the risk of overfitting the training data. The neural network's performance on an independent test set is the standard by which the network model must be judged.

Neural networks are a powerful addition to the arsenal of statistical decision-making methods available to researchers. Their usefulness in real applications should be compared to that of simpler models such as the discriminant function. Since training the neural network typically requires several hundred or more iterations of the data set, it is wise to examine whether the increased model complexity is justified by the improved fit to the data.

References

[1] H. Akaike, A new look at the statistical model identification, *IEEE Transactions on Automatic Control* **AC 19** (1974), 716–723.
[2] M.L. Astion and P. Wilding, The application of backpropagation neural networks to problems in pathology and laboratory medicine, *Archives of Pathology and Laboratory Medicine* **116** (1992), 995–1001.
[3] E.B. Baum and D. Haussler, What size net gives valid generalization?, *Neural Computation* **1** (1989), 151–160.
[4] G. Cybenko, Approximation by superposition of a sigmoidal function, *Mathematics of Control, Signals and Systems* **2** (1989), 303–314.
[5] D.B. Fogel, An information criterion for optimal neural network selection, *IEEE Transactions on Neural Networks* **2** (1991), 490–497.
[6] J. Hertz, A. Krogh and R.G. Palmer, *Introduction to the Theory of Neural Computation*, Sante Fe Institute in the Sciences of Complexity, Addison-Wesley, 1991.
[7] R.A. Jacobs, Increased rates of convergence through learning rate adaptation, *Neural Networks* **1** (1988), 295–307.
[8] A.H. Kramer and A. Sangiovanni-Vincentelli, Efficient parallel learning algorithms for neural networks, in: *Advances in Neural Information Processing Systems I*, D. Touretzky and M. Kaufmann, eds, 1989, pp. 40–48.
[9] I.D. Longstaff and J.F. Cross, A pattern recognition approach to understanding the multi-layer perceptron, *Pattern Recognition Letters* **5** (1987), 315–319.
[10] W.H. Press, B.P. Flannery, S.A. Teulosky and W.T. Vetterling, *Numerical Recipes*, Cambridge University Press, 1986.
[11] B.D. Ripley, Statistical aspects of neural networks, in: *Chaos and Networks – Statistical and Probabilistic Aspects*, O.E. Barndorff-Nielsen, D.R. Cox, J.L. Jensen and W.S. Kendall, eds, Chapman and Hall, 1993.
[12] F. Rosenblatt, *Principles of Neurodynamics*, Spartan, 1962.
[13] D.E. Rumelhart, J.L. McClelland and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, MIT Press, 1986.
[14] R.L. Watrous, Learning algorithms for connectionist networks: Applied gradient methods of nonlinear optimization, in: *IEEE First International Conference on Neural Networks* (M. Caudill, ed.) **2** (1987), 619–627.