Classification Using Multi-Layered Perceptrons

Selwyn Piramuthu*
Michael J. Shaw§*
James A. Gentry**
CMU-RI-TR-90-29

§Robotics Institute, Carnegie Mellon University; on leave from The Beckman Institute, University of Illinois at Urbana-Champaign
* The Beckman Institute and Department of Business Administration, University of Illinois at Urbana-Champaign
** The Department of Finance, University of Illinois

Contents

List of Tables

List of Figures

Abstract

There has been an increasing interest in the applicability of neural networks in disparate domains. In this paper, we describe the use of multi-layered perceptrons, a type of neural network topology, for financial classification problems, with promising results. Backpropagation, which is the learning algorithm most often used in multi-layered perceptrons, however, is inherently an inefficient search procedure. We present improved procedures which have much better convergence properties. Using several financial classification applications as examples, we show the efficacy of using multi-layered perceptrons with improved learning algorithms. The modified learning algorithms have better performance, in terms of classification/prediction accuracies, than the methods previously used in the literature, such as probit analysis and similarity-based learning techniques.

1 Introduction

The need for classification or separation of data consisting of patterns or examples into their respective categories is an ubiquitous problem in business applications. Classification problems arise in such situations as credit or loan evaluation (Carter and Cartlett, 1987; Orgler, 1970), bond rating (Gentry, Newbold, and Whitford, 1988), consumer choice (Currim, Meyer, and Le, 1988), tax planning (Michaelsen, 1984), and prediction of bankruptcy of firms (Gentry, Newbold, and Whitford, 1985), among others.

Statistical procedures such as multiple discriminant analysis (Abdel-Khalik and El-Sheshai, 1980), probit (Finney, 1971), logit, and regression have been widely used (Alterman, et al., 1981) in business for classification purposes. Parametric statistical methods require the data to have a specific distribution, which is usually Gaussian. In addition to the restriction on the distributions involved, missing (omitted) variables, multi-collinearity, autocorrelation, linear versus quadratic model assumptions, and qualitative variables could lead to problems with the estimated model when statistical methods are used. Inductive learning (Michalski, 1983) and learning by using artificial neural networks (Rumelhart, et al, 1986) are two recent developments in the AI domain that have potentially promising applications for classification in business applications. Researchers (Braun and Chandler, 1987; Messier and Hansen, 1988; Mingers, 1987; Shaw and Gentry, 1990) have studied various inductive learning algorithms along with appropriate statistical methods with results favoring inductive learning algorithms over statistical methods. Inductive learning methods have better knowledge representational structure in the sense that the decision tree that is induced can be used to derive (production) rules which can be implemented in an expert system, thus facilitating in automating the classification process.

Neural networks, which learn by modifying weights in the connections in the network, are potentially advantageous over both inductive learning and statistical methods. Characteristics of neural network are its inherent parallelism, and its tolerance to noise, which is achieved by distributing knowledge across the network (Matheus and Hohensee, 1987). Neural networks have incremental learning facility which eases the process of updating knowledge as new examples are added to the system. The topology of neural network that is most suitable for classification purposes is the multi-layered perceptron along with back-propagation algorithm, which is discussed in detail in

section 3. Back-propagation utilizes steepest-descent method for searching in the problem space. The method of steepest-descent is the most simplistic of all gradient descent methods, and does not take the curvature of the search space into consideration. In this paper, we present a modified back-propagation algorithm with the Newton-Raphson algorithm, which utilizes information on the curvature of the search space for efficient convergence to the final solution, for gradient search.

Comparing neural-net methods with other classification methods, a number of prior studies (Dutta and Shekhar 1988; Fisher and McKusick, 1989; Mooney, Shavlik, Towell, and Gove, 1989; Weiss and Kapouleas, 1989) have found that the back-propagation algorithm achieves higher asymptotic accuracy levels, is better able to handle noise, but requires a larger training data set. It is also commonly acknowledged that a major problem with the back-propagation algorithm is that it is too slow.

The main objectives of this paper are: (1) to develop modified learning algorithms for improveing the back-propagation procedure, and (2) to show the effectiveness of using neural-networks, specifically, multi-layered perceptrons, as a tool for classification problems, focussing specifically on financial management and credit analysis. As will be discussed in subsequent sections, the results are very encouraging, showing definite value in using neural-networks for financial classification applications such as loan evaluation and credit analysis.

When implemented in the back-propagation algorithm, the modified (Newton-Raphson) method performed consistently better than the regular back-propagation algorithm, in terms of efficiency and classification/prediction accuracy. The hybrid-method, a combination of the regular as well as the Newton-Raphson type algorithms, performed the best overall. In addition to back-propagation, we use two similarity-based learning algorithms and probit to analyze three real-world financial data sets. Overall, the back-propagation algorithms performed at least as good as the other methods used in the literature for analyzing financial classification data.

In the next section, we present a brief overview of various methods that have been used for financial classification. In section 3, we review the back-propagation algorithm as implemented in a multi-layered perceptron. The Newton-Raphson algorithm is discussed in section 4. The modified back-propagation with Newton-Raphson algorithm and corresponding gradient derivations are discussed in section 5. Further improvements in performance

of the algorithm are considered by combining the strengths of both Newton-Raphson and Steepest-Gradient methods as implemented in the Hybrid algorithm which is given in section 6. An simulation example using financial classification, comparing the performances of the three algorithms (SG, NR, and NR-SG) as incorporated in back-propagation algorithm in multi-layered perceptrons are given in section 7. In section 8, three real-world financial classification examples are used to compare the performances of neural-net method with two Similarity-based learning algorithms and Probit. Section 9 concludes with insights gained through this study and its contributions.

2 Background

Traditionally, researchers interested in financial classification applications have applied parametric methods such as multiple discriminant analysis (MDA), probit, logit, and regression. Most of these statistical methods assume the data being used to be distributed gaussian. In addition to assumptions of the distributions involved, statistical methods are restricted by the following potential problems. While using qualitative variables in probit, when the probit regression lines that correspond to the different values of the qualitative variables are not parallel, interpretation of any comparison between them is difficult (Finney, 1971). Omitted variables, unless their regression coefficients are zero, could be a cause for non-zero mean value of error terms in regression analysis which in turn could lead to erroneous results. Multicollinearity, a major problem when analyzing real-world data, arises due to inter-dependencies among attributes. Auto-correlation is due to correlations between residual or error terms of two or more observations. Finally, assumption of regression functions to be linear or quadratic might induce additional bias in estimating parameters.

Recent advancements in *artificial intelligence* have spurred interest both among practitioners and researchers to evaluate some of the methods that have been developed under the rubric of AI as compared to statistical methods. Mingers (1987) compared the performances of a modified version of ID3 (Quinlan, 1986) and multiple regression and concluded in favor of induction. Messier and Hansen (1988) used loan default and bankruptcy data to compare the effectiveness of ID3, discriminant analysis, individual judgements, and group judgements in discovering predictive knowledge structures. After

extensive analysis they concluded that (similarity-based) inductive learning methods may be useful for small highly-structured problem domains, and also for expert system development. An advantage of similarity-based learning is its representation of causal relationships which are more informative for identifying the relative importance of attributes that are used in the study.

Theoretically artificial neural network systems are capable of acquiring any arbitrary mapping function (Hornik, Stinchcombe, and White, 1989), by means of appropriate number of nodes and links among them, and therefore have no inherent representational or learning advantages or disadvantages as compared to traditional machine learning systems. Both back-propagation algorithm and induction are independent of assumptions of the distributions of the sample of training data that are used in the learning process. Most of the shortcomings that are associated with statistical analyses are hence of no concern while using either back-propagation or inductive learning algorithms.

In addition to similarity-based learning methods, artificial neural networks are being widely used for similar purposes. Neural networks have the following inherent advantages over inductive learning methods:

- *Incremental learning* - induction of decision trees (Quinlan, 1986), is done probabilistically based on the number of positive examples that are present at a given node, and as such necessitates a need for all the examples to be present simultaneously, whereas back-propagation does not have this restriction - examples are included and learning occurs in the network as and when new examples are added to the network. This facilitates the process of incremental learning in neural networks by learning to modify current knowledge incrementally as new examples are added to the system.

- *Tolerance to noise* - classification analyses using real-world data entails a certain amount of immunity toward noise present in the data. Neural networks are inherently noise tolerant due to its distributed representation where knowledge is represented across all the nodes in the network. Since a single piece of information is distributed and stored in an aggregate manner among different nodes in the network, noise in a processing unit does not affect the overall behavior of the network significantly. Similarly, when only partial or incomplete information is available, the network provides immunity by attempting to complete the respective patterns by utilizing available partial information.

- *Polythetic learning/interaction effects* - given a set of observations, information can be obtained by considering each of the variables that are included in the observations as well as by considering the effects of variables on one another in influencing the final decision. Interaction between variables thus play a crucial role in providing a more complete information in addition to those from the individual variables themselves. These interactive effects are naturally captured in neural networks by means of connections from different variables that end up at a single node at a higher node, thus aiding in an efficient use of the information that is present in the input variables. In the inductive learning process, for splitting at a node each of the variables are evaluated one at a time, thus ignoring the effects of variable combinations (to account for interactive effects) explicitly. This could lead to a loss of information that could have otherwise been retrieved from the examples. The process of considering all the variables together (*polythetic*) in back-propagation (Fisher and McKusick, 1989), as opposed to one at a time as in inductive learning, fosters an improvement in performance of the back-propagation algorithm over inductive learning algorithms.

- *Concept representation* - knowledge is represented as strengths of weights in links in a neural network thus leading to tolerance of noise, whereas in inductive learning, explicit rules are formed for a given class, which need not represent the actual state of affairs in the data set of interest. A single wrong example can lead to a misleading rule while using inductive learning, which is avoided in neural networks by diffusing the same across units in the network leading to severity of a lesser order. Whereas the induced rules are informative as to the knowledge they represent, it is very difficult, if not impossible, to derive meaning from the weights in the links in a neural network.

- *Finer granularity* - inductive learning algorithms are characterized by their coarser granularity due to the mode in which examples are covered. Neural networks, on the other hand, are relatively fine-grained due to distributed knowledge representation. The finer granularity of search in neural networks enables the search process to be focused more on the structure of the search space, thus leading to better learning.

As shown in figure 1, the classification process in statistical, inductive

learning, and neural network classifiers are done in two stages. In the statistical classifier, input values are used sequentially to evaluate the function value using the estimated parameter values. During the second stage, the best among different functional values is used to determine the (maximum likelihood) class of a given pattern. In the inductive learning classifier, input values are used sequentially to be passed on from the root of the decision tree toward its leaf nodes. The input class is deduced as per the leaf node to which the pattern belongs. In the neural network classifier, input values are propagated in parallel through the N_o input connections toward the output layer. For a given example, all the M units in the output layer might have non-zero values, and the output unit with the maximum value is selected to be the class of the given pattern.

insert figure 1 here

Artificial neural networks have shown promising results for a number of problem areas including content addressable memory, pattern recognition and association, category formation, speech production and global optimization (Kohonen 1984, Rumelhart, et al. 1986, Anderson 1977, Sejnowski and Rosenberg 1987, Hopfield and Tank 1986). It has also been tried with fairly good results in financial (Dutta and Shekhar 1988) as well as in manufacturing applications (Rangwala and Dornfeld, 1989).

Back-propagation algorithm uses an iterative method to feed the activation values from lower layers to the layers above and to propagate the error values backward from layers above to the next layer below. There are three broad classes of iterative methods that are used for optimization problems (Walsh, 1975): (1) quadratic methods, that use the first and second derivatives of the objective function in the neighborhood of the current trial solution, (2) linear methods that use first but not second derivatives, and (3) directional methods that use no derivatives.

Researchers (Becker and leCun, 1988; Fahlman, 1988; Parker, 1987; Watrous, 1988) have studied second order functions with promising results. The primary motivation behind this study is to improve the performance of the classical back-propagation algorithm by utilizing better gradient search methods than the simplistic steepest gradient method that is being currently used.

Recently, several researchers (Dutta and Shekhar, 1988; Fisher and McKusick, 1989; Mooney, Shavlik, Towell, and Gove, 1989; Weiss and Kapouleas, 1989) have studied the comparative performances of symbolic, connectionist, and statistical methods. Dutta and Shekhar (1988) used bond-rating data to compare the performances of back-propagation and regression, in which back-propagation out-performed regression in predicting bond ratings from a given set of financial ratios. Singleton and Surkan (1990) compared the classification performances of back-propagation and linear discriminant analysis on ratings by *Moody's* and *Standard and Poor's* indices and concluded in favor of back-propagation. Fisher and McKusick (1989) compared the performances of ID3 and Back-propagation in which Back-propagation performed consistently better in terms of asymptotic accuracy in classification. Mooney, Shavlik, Towell, and Gove (1989) used ID3, Perceptron, and Back-propagation in their study using four real-world data sets and concluded that Back-propagation performs better on noisy complex environments. Weiss and Kapouleas (1989) used four real-world data sets to compare the performances of statistical classifiers such as linear/quadratic, Bayes, nearest-neighbor, CART (Breiman, Friedman, Olshen, and Stone, 1984), ID3, and back-propagation and concluded that back-propagation performed at least as well as the other methods in terms of classification accuracy.

3 Back-propagation in Multilayer Perceptrons

Multi-layered perceptrons consist of a set of processing units (figure 2), with corresponding activation functions (a_i), links among the processing units, weights (W) corresponding to the links, and threshold (θ) values for each of the processing units. The processing units are arranged in layers and are connected to one another by restricted links. Each unit in a layer is connected to units in the immediate next and previous layers and not to any of the units in the same layer. Each unit acts as an integrator of the signals which it receives from the units in the layer which is immediately below the layer in which the unit of interest is located. The links between the units are all assigned weights as per the activation of the unit in the lower layer from which the link originates. Each of the units, except those in the input layer, are also assigned a threshold value (θ), which is used in deciding to propagate the units' activation forward. As shown in figure 2,

a processing unit j receives its input from all the units in the previous layer $(X_0, X_1, \cdots, X_{N-1})$, which is used to determine its output. The output from an unit (j) is a function (usually sigmoidal) of the difference between its threshold (θ_j) and the sum of weighted activations from previous layer.

insert figure 2 here

In situations where linearly separable functions are used, a single-layered network (with no hidden layers) is appropriate, and for non-linear functions, multi-layered networks are more appropriate. In single-layered networks, since the activation values of the output layer is influenced by only a linear combination of the inputs, only a few hyper-planes (as per the number of output units) can be represented. Single-layered networks are used in cases where a few hyper-planes are sufficient to separate inputs belonging to different categories. In cases where a convex region or any arbitrary function (Lippmann, 1987; Hornik, Stinchcombe, and White, 1989) is present, more than one layer is required to be able to separate examples belonging to different classes, by utilizing multiple-ordered effects of input values.

Back-propagation (Werbos, 1974) is one of the most widely used among the different neural-network algorithms that are used for classification purposes. Back-propagation is carried out in layered feed-forward networks with input and output nodes at the extremes (the back-propagation algorithm is given later in this section). Figure 3 in section 7 is an example of a multi-layered perceptron network. The input layer has 15 ($N_o = 15$) units, the (h = 1) hidden layer has 10 ($N_1 = 10$) units, and the output layer has 5 (M = 5) units. All the 15 input units are connected to all 10 units in the first hidden layer, which are all in turn connected to all the 5 units in the output layer. Activations from input layer is passed on to the first layer, which then passes on its activations to the output layer. The output layer passes on the error backward to the first hidden layer, and the weights in links between the first hidden layer and the output layer are modified as per the back-propagation algorithm that is given below.

Back-Propagation (BP) Algorithm

1. *Initialize* - Set number of units in input (N_o), output (M), and hidden (N_k; k=1 .. h) layers, where h is the number of hidden layers. Specify links between units with no links between two units in the same layer. Set random values for weights in all the bias levels for all the units (except those in the input layer).

2. *Input* - Present input/output vectors $i_1, \ldots, i_{N-1}, o_1, \ldots, o_{M-1}$.

3. *Update weights:* $w_{ij}(n+1) = w_{ij}(n) + \eta\delta_j a_i + \alpha(w_{ij}(n) - w_{ij}(n-1))$ where $w_{ij}(n+1)$ is the weight in the link from unit i to unit j in the n^{th} iteration (epoch), η the learning rate, and α the momentum term. The weights during the initial pass are set at random.

4. *Calculate actual outputs*:
   for hidden layers ($x_i = i_i$ if unit i is an input unit):

$$a_j = f(\sum_{i=0}^{N_k-1} w_{ij}x_i - \theta_j), 0 \leq j \leq N_k$$

   for output layer:

$$a_j = f(\sum_{i=0}^{M-1} w_{ij}x_i - \theta_j), 0 \leq j \leq N_h$$

   where θ_j is the threshold in the j^{th} unit in the layer under consideration.

5. *Update δ-values*:
   if unit(j) is an output unit:

$$\delta_j = (o_j - a_j)f'(net_j)$$

   if unit(j) is a hidden unit:

$$\delta_j = f'(net_j)\sum_k \delta_k w_{jk}$$

6. *Repeat*: Go to step 2.

Since there is no set prescribed number of hidden units or hidden layers that need to be used in a back-propagation network, a certain amount of experimenting with different number of hidden units are required before settling on a final network configuration. The number of input units is set to be equal to the number of variables of interest in the input data. Since the output units are binary, the number of units in the output layer are chosen as per the number of classes in the data (for example, 1 unit which takes on values 0 or 1 if there are 2 classes). In this study, the number of hidden units (with 1 hidden layer) are taken to be about half the total number of input and output units.

Initially, weights to the links are assigned randomly and the resulting (actual) output is compared with the target output provided by the teacher (steps 1-5). The difference between the actual and the target output is the error, which is summed over all the output units. Since the purpose here is to minimize the error that results from the weights that are assigned to the links, the objective function for this optimization problem is taken to be the total sums-of-squares (tss) of the differences between the actual and the target values for the output nodes. Weights in the links are then modified by means of a function that takes the gradient of the objective function into account (step 3). Steps 1-6 are repeated until the pre-specified tss value (ecrit) is reached or the pre-specified number of epochs (iterations) are reached. Step 5 incorporates the gradient search information (as derived in section 4) by means of δ, the step length in the direction of descent. Since this is a gradient search method, it has all the pitfalls that are associated with gradient descent methods such as finding a local minima or a saddle point. The Newton-Raphson algorithm discussed in the next section avoids saddle points although local minima would still remain a problem.

The number of nodes and the connections between them are configured initially and are never modified unless it is absolutely necessary due to improved performance otherwise. The patterns learned are stored in the same set of nodes overlapping one another, in a sense, as weights in the links and also in the threshold values that are associated with each node. Hence the time required to produce output patterns is independent of the number of patterns that are stored in the network.

4 Newton-Raphson (NR) Algorithm

For non-linear functional optimization problems, gradient descent methods are used for finding the best (if possible, optimal) solution by searching (reverse hill-climbing) the solution space iteratively. Initially, an arbitrary point in the search space is chosen, from which the final solution is approached iteratively by descending along the direction which provides improved solutions. The Newton-Raphson algorithm is one of the several classes of methods that are based on a quadratic model of the objective function, in contrast to the scaled linear model function of the classical steepest-gradient method. There are two major justifications for choosing a quadratic model: its simplicity and, more importantly, the success and efficiency in practice of methods based on it (Gill , Murray, and Wright, 1981).

A quadratic model of the objective function is obtained by taking the first three terms of the Taylor-series expansion about the current point.

$$f(x_k + p) \approx f(x_k) + g_k^T p + \frac{1}{2} p^T G_k p$$

where, x_k is the current estimate of x^*, the minima; p is the direction of search; $g_k (= g(x_k))$ is the first derivative of f, and $G_k (= G(x_k))$ is the second derivative.

The quadratic function is then formulated in terms of the step to the minimum (p) rather than the predicted minimum itself. The minimum of the above function is achieved if p_k is a minimum of the quadratic function:

$$\Phi = g_k^T p + \frac{1}{2} p^T G_k p.$$

Suppose that Φ takes its minimum value corresponding to $p = p_k$, then, $\nabla_{p_k} \Phi = 0$. i.e., $G_k p_k + g_k = 0$, which yields $p_k = -G_k^{-1} g_k$. The Newton-Raphson method uses p_k as the current direction of search which is updated iteratively.

In the simplest problem of unconstrained minimization of the univariate function f:

$$\min_{x \epsilon R^1} f(x)$$

if a function is everywhere twice-continuously differentiable, and a local minima of the function exists at a finite point x*, then the following two necessary and sufficient conditions for an unconstrained minimum must hold at x*:

1.

$$f'(x^*) = 0$$

and

2.

$$f''(x^*) \geq 0$$

The local convergence properties of Newton-Raphson method make it an exceptionally attractive algorithm for unconstrained minimization. A further benefit of the availability of the second derivatives is that the sufficient conditions for optimality can be verified as stated above.

The Newton-Raphson algorithm can be summarized as follows:

1. *initialize* - for any starting point x_k, Taylor-series expansion (first 3 terms) about the current point:

$$f(x_k + p) \approx f(x_k) + g_k^T p + \frac{1}{2} p^T G_k p$$

2. *minimize*:

$$\Phi = g_k^T p + \frac{1}{2} p^T G_k p$$

3. *obtain gradient*

$$\nabla_{p_k} \Phi = 0, \ \textit{giving} \ \ G_k p_k + g_k = 0$$

4. *search in direction*: $p_k = -G_k^{-1} g_k$.

Gill, Murray, and Wright (1981, p286) rank different methods for solving unconstrained problems based on the confidence that can be placed in the methods' success in finding an acceptable solution. Newton-type algorithms with second derivatives is ranked first, followed by Newton-type without second derivatives, Quasi-Newton with first derivatives, Quasi-Newton without first derivatives, Conjugate-Gradient type with first derivatives, conjugate-gradient without first derivatives, and polytope. The superiority of the Newton-Raphson type method reflects the use of the second order properties

of the function to improve algorithmic efficiency as well as to provide qualitative information about the computed solution (Gill, Murray, and Wright, 1981, p320). An additional benefit using the Newton-Raphson type method is that there are certain problem classes (eg., those with saddle points) where only a Newton-type method is assured of being able to proceed.

Local minima, which is a problem with all gradient search methods, is still a problem with the Newton-Raphson method. But, it can still be used for finding local minima when there are saddle points, or when the problem is known to be convex, since the local minimum must then be a global minimum. Many real-world problems are not convex, but one may still be content with a local minimum because the non-convex elements in the problem are in the nature of small perturbations (which are probably saddle points), so that it seems unlikely that they could introduce unwanted local minima or, because the only practical alternative to a local minima is a completely arbitrary solution.

5 Modified BP with NR type Algorithm

The back-propagation algorithm utilizing the steepest-gradient descent method is given in section 2. In the steepest-gradient method (Rumelhart, et al., 1986), the objective (error) function that is minimized is

$$E = \sum_p E_p = \frac{1}{2} \sum_p \sum_j (o_{pj} - a_{pj})^2 \tag{1}$$

where, p corresponds to the pattern that is currently under consideration, o_{pj} is the target output value of the j^{th} unit in the output layer when pattern p is input, a_{pj} is the actual output value of the j^{th} unit in the output layer when pattern p is input, and E is the sum of errors over all the patterns that are input. Once the threshold(θ) values are set, the θ terms can be ignored for further analysis. For unit j receiving input from unit i's in the lower layer,

$$net_j = \sum_i w_{ij} a_i \tag{2}$$

where,

$$a_i = f(net_i) = \frac{1}{1 + e^{-net_i}} \tag{3}$$

a sigmoid (squashing) function. Since the change in weight in a link is directly (negatively) proportional to the rate of change of the error function with respect to a change in weight in the link of interest,

$$\Delta_p w_{ij} \propto -\frac{\partial E_p}{\partial w_{ij}}$$

By chain rule, the derivative can be written as the product of two parts: the derivative of the error with respect to the output of the unit times the derivative of the output with respect to the weight.

$$\frac{\partial E_p}{\partial w_{ij}} = \frac{\partial E_p}{\partial net_{pj}} \frac{\partial net_{pj}}{\partial w_{ij}} \tag{4}$$

from equation 2, we get

$$\frac{\partial net_{pj}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_k w_{kj} a_{pk} = a_{pi} \tag{5}$$

define

$$\delta_{pj} = -\frac{\partial E_p}{\partial net_{pj}} = -\frac{\partial E_p}{\partial a_{pj}} \frac{\partial a_{pj}}{\partial net_{pj}} \tag{6}$$

from equation 3,

$$\frac{\partial a_{pj}}{\partial net_{pj}} = f'(net_{pj}) \tag{7}$$

which is the derivative of the squashing function f for the j^{th} unit, evaluated at the net input net_{pj} to that unit.

If j is an unit in the output layer of the network, from equation 1,

$$\frac{\partial E_p}{\partial a_{pj}} = -(o_{pj} - a_{pj}) \tag{8}$$

from equations 6, 7, and 8, we get

$$\delta_{pj} = (o_{pj} - a_{pj}) f'(net_{pj}) \tag{9}$$

In the proposed method, the idea of optimizing a quadratic function as in the Newton- Raphson method is utilized. The original objective function, in the classical steepest gradient method, of minimizing the squares of the differences between the actual and the desired output values summed over the

output units and all pairs of input/output vectors is replaced by an equivalent exponential function. This is done in order to allow for the second derivative to be derived from the objective function. The direction of descent which minimizes $(o_j - a_j)^2$ is equivalent to the direction of descent which minimizes $e^{(o_j-a_j)^2}$, and the transformation is justified since both the exponential of a non-negative function and the function itself are monotonically increasing. By taking the exponential, the granularity of search is increased as the step-size is increased.

The function to be minimized (ignoring the subscript p) is:

$$f(a) = e^{(o_j-a_j)^2} \tag{10}$$

where o_j and a_j are the target and actual outputs (active values) of the j^{th} unit in the output layer. Instead of taking the second order gradient direction p_k as in previous attempts using second-order methods for back-propagation, we use the gradient $G_k p_k + g_k$. This avoids the inversion of the Hessian matrix G_k, although it is only a scalar because we are only dealing with one output unit at a time. The corresponding gradient, with respect to a_j, of the quadratic function, as per the Taylor's series expansion in the Newton-Raphson algorithm can be derived as follows:

$$g_j = -2(o_j - a_j)e^{(o_j-a_j)^2}$$

$$G_j = 2e^{(o_j-a_j)^2} + 4(o_j - a_j)^2 e^{(o_j-a_j)^2}$$

from which we derive,

$$\Phi(a_j) = 4(o_j - a_j)^3 e^{(o_j-a_j)^2} \tag{11}$$

substituting equation 11 for the gradient (equation 8) and equation 7 in equation 6, we get

$$\delta_j = 4(o_j - a_j)^3 e^{(o_j-a_j)^2} f'(net_j) \tag{12}$$

which is the corresponding modification in step 5 of the back-propagation algorithm (section 3) if unit j is an output unit.

where $f'(net_j)$ is the derivative of the activation function with respect to a change in the net input to unit j. If unit j is a hidden unit, the modified delta values are propagated from the output units downward to the hidden

units. Hence the activations of the hidden units are indirectly affected by the modified delta values of units in the output layer. Since the second order derivative of the objective function is taken into consideration here, the algorithm is insensitive to saddle points in the solution space. Convergence is also much faster since a quadratic function (as opposed to a linear function as in the steepest-gradient method) is utilized in the solution procedure.

6 The Hybrid (NR-SG) Algorithm

By using the second-order gradient, the NR algorithm search through the weight space would demonstrate the following behavior: During the initial stages in the search process, when the differences between the actual and target classes are large, the NR algorithm tends to take larger step sizes due to the exponential transformation. This could lead to undesirable effects such as thrashing since the curvature of the search space is not effectively taken into consideration by the NR algorithm unless the starting point is closer to the optimal solution. Once the search reaches the proximity of the optimal solution, the step size taken by the NR algorithm decreases drastically and is thus more effective in converging to the final solution faster. The motivation for using the hybrid (NR-SG) algorithm is the relative good performances of SG and NR algorithms farther and closer to the optimal solution respectively. Both steepest-gradient descent and Newton-Raphson methods were incorporated in this algorithm. The Steepest-Gradient (SG) method is used to get closer to the final solution and then the Newton-Raphson (NR) method is initiated from that point to converge to the final solution. The final solution thus reached would be at least as good as that which is reached by the NR or SG method alone, and the convergence is also faster in the hybrid method than in the SG and NR methods alone. The hybrid algorithm is as follows:

1. *initialize* - steps 1 and 2 as in the back-propagation algorithm as given in section 3.

2. *calculate the initial tss value*

$$tss_{initial} = \sum_{p} \sum_{j} (o_{pj} - a_{pj})^2$$

where, p is over all the input patterns, and j is over all the units in the output layer.

3. *SG algorithm* - follow the back-propagation algorithm as given in section 2 until

$$tss = (tss_{initial} - 0.04) * 0.8$$

4. *NR algorithm* - apply NR algorithm as given in section 5 until the tss value becomes less than or equal to 0.04 (the pre-specified ecrit value).

In this study, steepest-descent algorithm was used during the initial period until 80 percent of the difference between the final tss (total sums-of-squares for all the input patterns - tss is the sums of squares of the differences between the actual and the target activation values of the units in the output layer) value (ecrit, was set at 0.04) and the initial tss value, was reached. Hence, by using the NR method only for the last 20 percent of the values of tss a better performance of the NR algorithm is guaranteed than otherwise.

7 A Financial Classification Example

For financial classification, the input patterns are taken to be the independent variables, and the output patterns are the dependent variables. Learning in the network takes place by encoding the input patterns (independent variables) in the hidden layer(s), which is then decoded to map to the output patterns (dependent variables). In this section, we use a loan evaluation data set to go through the mapping of classification problems on to multi-layered perceptrons. A brief description of the loan evaluation data is given in appendix-B, and the data is used for detailed analyses in the next section. In addition to the 12 continuous (cash flow components) variables, 3 qualitative components (nominal variables) were also included in this data set. The three qualitative variables are guarantee, collateral, and secured - both guarantee and collateral are boolean variables which take values of either yes or no, and secured taked values 0 or 1 depending on whether a firm is secured or not respectively. Each of the 15 independent variables are assigned a separate input unit in the multi-layered perceptron network. Since there are 5 classes, we have 5 output units, one for each of the classes. The output unit with the highest activation value is taken to represent the class

of a given example. The number of hidden units in the only hidden layer used is taken to be 10 (half of the total number of input and output units). The network is given in figure 3, and the weights in the links and the bias values corresponding to the hidden and output layers are given in table-1. Convergence of the network for SG, NR, and NR-SG are given in figure 4. As can be seen from figure 4, for the SG algorithm, the tss value remains at around 7 for a wide range of the number of epochs, and still did not converge after 30,000 epochs.

insert figures 3 and 4 here

unit #	15 (0.74)	16 (2.8)	17 (5.99)	18 (-7.93)	19 (-1.05)	20 (-1.85)	21 (-1.73)	22 (6.29)	23 (0.65)	24 (5.82)
0	-0.83	-10.56	2.71	-11.23	-20.08	-4.66	10.65	-8.01	-10.27	-0.63
1	-7.99	-2.49	13.49	14.58	8.74	-6.63	9.24	-8.88	-3.86	4.35
2	-8.66	-2.41	-2.33	-4.9	24.17	-7.26	-8.6	-13.05	6.94	1.05
3	2.75	-0.76	-11.22	12.46	4.57	-3.35	-16.47	-3.07	15	2.78
4	0.52	10.23	12.6	-5.83	-2.9	5.1	7.44	0.04	-6.29	5.28
5	-7.71	-7.39	-4.97	15.25	-9.95	-2.9	1.3	5.25	-3.87	-0.62
6	-0.97	-7.82	7.88	12.84	-16.53	-6.39	21.26	-2.07	-0.87	-0.11
7	-11.69	10.72	-2.16	1.47	8.35	1.94	-11.42	-4.5	1.03	2.76
8	-19.17	15.03	2.29	-11.4	-11.25	4.83	-3.31	11.56	-25.58	9.27
9	1.68	-8.32	0.1	-32.9	-1.95	-4.48	-2.27	-17.76	-1.5	3.65
10	4.92	-5.61	-17.68	-11.18	-1.13	-0.48	-21.14	4.94	8.5	-1.12
11	1.32	-1.86	-14.02	21.45	-0.11	-4.29	-14.45	17.23	0.74	4.59
12	-4.85	0.55	-11.67	-8.14	-6.4	0.31	-1.31	5.92	-9.88	-5.83
13	-3.22	-3.82	-3.11	-9.3	12.2	-2.76	-2.6	4.04	-7.35	-8.04
14	-2.56	4.41	-1.61	0.3	-6.88	0.9	-0.75	-2.23	1.67	5.07
25 (-5.95)	-7.26	-8.36	12.69	5.22	-18.92	-2.37	10.58	-1.43	-5.01	-2.43
26 (0.38)	8.63	12.85	-9.08	-12.48	-9.67	-0.93	-2.27	-23.56	1.08	5.71
27 (-1.53)	-7.68	-4.65	-18.2	9.93	7.65	-3.47	-9.12	-7.53	-9.32	6.17
28 (-3.47)	-0.2	7.04	-2.21	-8.38	-4.44	5.83	-3.34	8.45	-3.31	-2.36
29 (-3.99)	-1.38	-1.45	1.48	-3.04	2.45	-1.3	-1.38	1.47	5.43	-19.52

Table-1: Weights and bias values using loan evaluation data

For the network, the learning rate parameter (η) was set at 0.25. The momentum term (α) was set at 0.9 to allow for faster learning by filtering out high curvatures and by allowing the effective weight steps to be larger. The tss value to be reached for convergence (ecrit) was randomly assigned a value of 0.04 as per Rumelhart et al. (1986). The input units are numbered from 0 through 14, hidden units numbered from 15 through 24, and the output units from 25 through 29. The bias terms for both hidden units and output units are given in brackets next to the appropriate units. Both continuous as well as nominal variables were used as input without any additional pre-processing such as by using dummy variables.

8 Applications

Three real-world financial risk classification data sets are used to compare the performances of different algorithms under different input conditions. Data sets from Abdel-Khalik and El-Sheshai (1980), Gentry, et al. (1985), and a loan evaluation data are used to compare the relative performances of different algorithms under varying conditions of noise present in real-world data. The different risk classification data sets (for evaluating loan payments, bankruptcy prediction and loan evaluation respectively) were used due to their differences in data structure. The Abdel-Khalik and El-Sheshai (1980) data consists of ratios and trends, the bankruptcy data set consists of 9 ratios, and the loan evaluation data set consists of 12 real-valued continuous variables and 3 nominal variables. Also the number of classes were 2, 2, and 5 in the loan payments, bankruptcy and loan evaluation data sets respectively.

The performances of different methods are compared with respect to their prediction/classification accuracies, and the time and number of epochs taken by each to converge to a pre-specified tss value (= 0.04). In addition to back-propagation, performances of ID3 and NEWQ - two similarity-based learning algorithms, and Probit (Finney, 1971), a standard statistical procedure are also studied. ID3 and NEWQ are briefly discussed in appendix-A, and a brief description of the three real-world data are given in appendix-B.

8.1 Default Classification

For the Abdel-Khalik and El-Sheshai (1980) data set, the learning rate parameter (η) was set at 0.25, momentum term (α) was set to 0.9, and the final convergence value for tss (ecrit) was set at 0.04. The network consisted of 18 input units (corresponding to the 18 input attributes), 1 output unit and a hidden layer with 10 hidden units.

Mooney et al. (1989) used 10 % of the total number of input and output units as the number of units in the hidden layer, which they found empirically to work well. Fisher and McKusick (1989) tried 0, 10, and 20 hidden units using their data sets from three domains. They reported results only using 10 hidden units which was the best over all domains. In Mooney et al. and Fisher and McKusick's study, the input variables were all represented in binary form. Effects on convergence using different number of hidden units using this data is given in figure 5. From figure 5, the convergence remains stable around 150 epochs over a wide range of number of hidden units. So, we decided on 10 hidden units, which is about half of the total number of input and output units. Ten separate simulation runs were done for all three back-propagation algorithms and the average values are reported in table-2.

---

insert figure 5 here

In addition to SG, NR, and NR-SG, ID3, NEWQ, and Probit analyses were also done using the same data set for comparison purposes. The 32 examples were used for training the network, or decision tree, or to obtain the discriminant function as the case may be. There was a significant decrease in the number of epochs required as well as the time taken for NR and NR-SG as compared to those for SG. Simulations were stopped either when the *tss* value was less than 0.04 or when the number of epochs reached 30,000. Even after 30,000 epochs, the tss for SG remained at 1.001 (on an average), and presumably would have taken many more epochs to converge to 0.04. The classification accuracy was a 100 percent in all the neural-network algorithms, ID3, and NEWQ. The classification accuracy of Probit was just 78.1%, which probably is due to the fact that Probit uses just a single hyper-plane (which severely limits its ability to separate the examples belonging to different classes effectively) to separate the two classes, whereas this is not the case

with the other methods.

Surprisingly, both classification and prediction results for SG, NR, NR-SG, ID3, and NEWQ turned out to be the same, possibly due to the size of the data set. This could also be because the points corresponding to the examples in the example space are possibly scattered such that only 14 of the 16 testing examples fall under hyper-planes belonging to specific classes. The results using ID3 are similar to those reported in Messier and Hansen (1988). This prompted us to utilize other data sets, with different characteristics, in this study for more conclusive results.

Although different machines were used for running the different programs, run times are given in table-2 for comparing those of the back-propagation algorithms. Since the different programs were run on different machines under different environments (SG, NR, and NR-SG on a Convex-C220 under UNIX, ID3 and NEWQ on IBM-PC/AT-286 under DOS, and Probit on IBM-3081 under VM/CMS), the run-times cannot be compared directly. The NR-type algorithms performed much better than SG in terms of speed of convergence. Due to the inherent parallelism in the back-propagation algorithm, speed-ups in multiple orders of magnitude can be achieved by vectorizing the code.

	SG	NR	NR-SG	ID3	NEWQ	PROBIT
epochs	30000	151.7	86.2	N/A	N/A	N/A
time(sec.)	9024	86.2	50.4	46*	12*	11**
tss	1.001	0.04	0.04	N/A	N/A	N/A
classification(%)	100	100	100	100	100	78.1
prediction(%)	87.5	87.5	87.5	87.5	87.5	37.5

(* both ID3 and NEWQ were run using IBM-PC/AT-286)
(** Probit was run using an IBM 3081)

Table-2: Summary of results using Abdel-Khalik, et al. data

8.2 Bankruptcy Prediction

With the growth in the number of bankruptcy cases, it is becoming increasingly important for financial firms to be able to predict whether a client firm will become bankrupt or not in the imminent future. Hence, it is of vital importance to study and to improve the decision process involved in bankruptcy prediction. As was done in Gentry, et al.(1985), the bankruptcy

data set was split into a training and a testing set. The training set consisted of 58 randomly selected examples and testing set consisted of the 46 remaining examples. Ten different random sets of training and testing data were used in the study, and the average of the ten different analyses are summarized in table-3. The individual classification/prediction values are given in table-4.

	SG	NR	NR-SG	ID3	NEWQ	PROBIT
epochs	30000	3086	1842	N/A	N/A	N/A
time(sec.)	10045	929	429	52*	17.8*	9.9**
tss	3.184	.04	.04	N/A	N/A	N/A
classification(%)	100	100	100	100	95.5	84.3
prediction(%)	67.4	69.6	69.6	56.7	62.3	54.3
total(%)	85.6	86.5	86.5	80.9	80.8	71.1

(*both ID3 and NEWQ were run using IBM-PC/AT-286)
(** Probit was run using IBM 3081)

Table-3: Summary of results using bankruptcy data

The learning rate parameter (η) was set at 0.25, momentum term (α) was set at 0.9, and the ecrit value was set at 0.04. The back-propagation network was made up of 9 input units corresponding to the 9 input attributes, 1 (binary) output unit corresponding to bankrupt and not-bankrupt firms, and a hidden layer with 5 hidden units (half of the total number of input and output units).

Since there is no set means to determine the number of hidden units to be used, we experimented with 1 through 10 hidden units in the network using the NR method. From the results of these runs, given in figure 6, we selected 5 hidden units, which required the least number of epochs to converge and is half the total number of input and output units. The network did not converge with 1 through 4 hidden units even after 30,000 epochs, and was very unstable probably due to the input variables being continuous rather than nominal. Results using 5 through 10 hidden units are given in figure 6.

insert figure 6 here

	SG	NR	NR-SG	ID3	NEWQ	PROBIT
classification(%)						
1	100	100	100	100	93.1	86.2
2	100	100	100	100	100	84.5
3	100	100	100	100	98.3	82.8
4	100	100	100	100	93.1	82.8
5	100	100	100	100	96.6	86.2
6	100	100	100	100	93.1	77.6
7	100	100	100	100	94.8	86.2
8	100	100	100	100	94.8	82.8
9	100	100	100	100	98.3	84.5
10	100	100	100	100	93.1	89.7
prediction(%)						
1	65.2	65.2	65.2	43.5	58.7	52.2
2	71.7	71.7	71.7	52.2	58.7	56.5
3	67.4	69.6	69.6	56.5	52.2	54.3
4	69.6	71.7	71.7	63.0	67.4	60.9
5	69.6	69.6	69.6	69.6	65.2	54.3
6	63.0	65.2	65.2	56.5	65.2	50.0
7	65.2	65.2	65.2	63.0	71.7	58.7
8	52.2	56.5	56.5	45.7	56.5	47.8
9	76.1	82.6	82.6	69.6	60.9	56.5
10	73.9	78.3	78.3	47.8	65.2	52.2

Table-4: Individual classification/prediction results using bankruptcy data

As can be seen from table-4, prediction results for both NR and NR-SG were the same throughout, although slightly better than SG in some cases. Both training (classification) and testing (prediction) were done using hold-out samples. Again, NR and NR-SG type algorithms performed significantly better than SG-type algorithm in terms of the number of epochs and also run-times for comparable classification/prediction accuracies. Although the total performance of ID3 and NEWQ were almost the same, the classification accuracy of ID3 was better than those of NEWQ, and the prediction accuracy using NEWQ was better than those using ID3. The overall performance of Probit was the worst of all the methods, although the prediction accuracy using Probit is comparable to those of NR and NR-SG type algorithms as implemented in back-propagation.

8.3 Loan Evaluation

The purpose of the loan evaluation process is to evaluate the credit-worthiness of firms, as a measure of their abilities to repay loans. As noted by Shaw and Gentry (1990), in addition to information available from financial statements, qualitative information play a vital role in the loan evaluation process. The loan evaluation data set was partitioned into a training and a testing set, each consisting of 50 examples. NEWQ was not used since NEWQ only uses binary class values and the data has 5 classes. For the same reason, *polytomous probit* was used instead of the regular binary probit in which the dependent variable can take on only binary values. As was done for bankruptcy data, ten different random sets of training and testing examples were used, and the average values from ten different analyses are summarized in table-5. As before, the individual values for classification/prediction are given in table-6.

	SG	NR	NR-SG	ID3	PROBIT
epochs	30000	1880	1276	N/A	N/A
time(sec.)	13764	1218	634	73	16
tss	3.989	0.04	0.04	N/A	N/A
classification(%)	100	100	100	100	58.2
prediction(%)	53.4	58.1	60	38	49.8
total(%)	76.7	79.1	80	69	54

Table-5: Summary of results using loan evaluation data

As before, η was set at 0.25, α was set at 0.9, and ecrit at 0.04. The multi-layered perceptron network for this data consisted of 15 input units, 10 hidden units in a hidden layer, and 5 output units corresponding to the five risk classification categories. The classification performance using probit was much lower than those using other (similarity-based learning or back-propagation) methods. Polytomous probit, which was used for this data set, classified all the examples to be in class 3 (there were 54 examples in class 3). Hence, the percent classification/prediction using probit is proportional to the number of examples belonging to class 3 which happened to be used in the training and testing sets respectively. This reason could be attributed to the poor performance of ID3 as compared to probit.

	SG	NR	NR-SG	ID3	PROBIT
classification(%)					
1	100	100	100	100	68
2	100	100	100	100	52
3	100	100	100	100	56
4	100	100	100	100	66
5	100	100	100	100	60
6	100	100	100	100	58
7	100	100	100	100	54
8	100	100	100	100	46
9	100	100	100	100	60
10	100	100	100	100	62
prediction(%)					
1	44	53	56	40	40
2	58	62	64	34	56
3	60	64	64	40	52
4	44	56	62	38	42
5	50	60	62	48	48
6	50	54	60	36	50
7	58	58	58	36	54
8	64	68	68	30	62
9	52	52	52	46	48
10	54	54	54	32	46

Table-6: Individual classification/prediction results using loan evaluation data

8.4 Discussion

The NR algorithm performed consistently better than SG for all three data sets. This is due to an increase in the granularity of the search process by means of the exponential transformation of the objective function, and also due to the second-order gradient search procedure. The increase in granularity increases the step-size taken during search through the solution space, hence increasing the speed of convergence of NR algorithm. The second-order gradient search process in NR utilizes information about curvatures in the search space thus modifying the search process appropriately leading to faster

convergence to the final value, whereas the SG method did not converge to the final solution. The SG algorithm had problems converging because once in the vicinity of the final solution the process started vacilating between a few values resulting in a movement further away from the final solution.

The hybrid algorithm performed even better than NR since it had the advantages of both NR and SG - it moved toward the final solution slowly while following the SG algorithm, and once in the vicinity of the final solution, it utilized the curvature information to converge to the final solution by following the NR algorithm.

Becker and LeCun (1988), Fahlman (1988), Parker (1987), and Waltrous (1988), among others, have suggested the use of second-order gradient methods for accelerating the back-propagation procedure. However these methods rely on matrix inversion and thus are more costly computationally. The NR method uses an exponential objective function along with modifications to the SG algorithm avoiding matrix inversion at the same time taking advantage of second-order process, thus making the computation tasks in the learning process easier.

One of the major problems with back-propagation algorithm is that it is very slow to converge. This could be remedied either by improving the algorithm itself and/or utilizing parallel computer architecture for faster processing. In this paper, we have attempted to inprove the performance of the algorithm. Even with the improved learning procedure, the computation performances of multi-layered perceptrons in our study are still slow for three reasons - the computation is simulated using serial processing computers, it takes many iterations of adjustment by each unit before the whole network reaches equilibrium, and it takes a large number of searching steps to find the right set of connection weights. To speed up the training time parallel computers are necessary. Hinton (1985), Zhang, et al. (1989), and Deprit (1989) describe using parallel processing computers and the Connection Machine to implement back-propagation algorithm for multi-layered perceptrons. Kung, Vlontos, and Hwang (1990) describe a VLSI architecture for implementing multi-layered perceptrons using a programmable systolic array. Because of the parallel processing nature of neural-network computing, the use of parallel computers is necessary to take advantage of the inherent parallelism. Similar observations can be made of the information processing characteristics of human brain. Even though the individual neurons are slow, the brain processes information expeditiously through massive parallelism. Re-

cent technological advances in VLSI and computer aided design have made it possible to build massively parallel computers, making the neural-network computing method described in this paper practical for commercial applications.

9 Review of Results and Conclusions

The modified back-propagation algorithms (NR and NR-SG) performed better than the regular back-propagation (SG) algorithm in terms of prediction accuracy and speed of convergence due to the exponential transformations and second-order gradient search. The NR-SG algorithm, due to its combination of NR and SG algorithms when they are at their best, performed the best overall even among the back-propagation algorithms. Even while taking advantage of the benefits of a second-order method, the computational process was simplified, in NR and NR-SG, by avoiding matrix inversions in the process.

Back-propagation algorithm in a multi-layered perceptron is versatile and can be used for learning ratios, continuous, as well as nominal variables with any number of classes resulting in lesser degradation in performance than inductive learning algorithms and probit. For continuous variables, ID3 has problems finding the best split at a node due to the problem of combinatorial explosion as the range of values that a continuous variable takes increases. Since the number of discriminant functions required increases with the number of classes, probit analysis becomes extremely difficult as the number of classes increases.

Once implemented (learning occurs), the process of prediction of newer examples is faster in back-propagation as compared to other methods since speed of prediction using back-propagation does not depend on the number of patterns that are stored in multi-layered perceptrons. Time required for prediction in multi-layered perceptrons is just the time required for activations to traverse from the input to the output nodes, which is negligible.

Overall, the back-propagation algorithms performed better than inductive learning algorithms and Probit in terms of both prediction and classification accuracies. One of the advantages of using back-propagation in a multi-layered perceptron is that the network can be configured so as to increase the performance, which is not possible with inductive learning or probit.

Also, incremental learning ability of back-propagation facilitates knowledge acquisition process where new knowledge can be added to the system as they are obtained.

Although the run times taken by the back-propagation algorithms are more than those taken by the similarity-based learning algorithms and Probit, by hard-wiring the algorithm by means of VLSI circuits, the time taken by the back-propagation algorithms can be made at least comparably faster as those of similarity-based learning algorithms and probit. Currently, there are VLSI implementations of neural networks such as Intel's AK107, Fujitsu's MB4442, NEC's UPD7281, among others, in addition to implementations in connection machines (Hinton 1985; Zhang et al. 1989; Deprit 1989).

Probit is inefficient relative to back-propagation and inductive learning algorithms since it utilizes only one hyperplane to separate two classes, whereas the former two methods utilize as many hyper-planes as is necessary. In back-propagation using multi-layered perceptrons, the hidden layer aids in avoiding the problem of forming linear hyperplanes.

Both back-propagation and ID3 algorithms do not assume any distributions that the input data should possess, unlike Probit. Since Probit depends on the distribution of the input data for analyses, violation of the same could lead to gross inaccuracies in the results that are obtained.

Although Baum and Haussler (1989) formulate a lower bound on the number of examples that are required for a given prediction accuracy, the small size of our data sets precluded us from applying the same in this study. Researchers (Ahmad, 1988; Kung and Hwang, 1988) have resorted to experimentation with different numbers of hidden units before resorting to final analysis. To this end, future research should be directed toward studying the number of hidden units that are both *sufficient* and *feasible* given a certain input-output data set.

References

[1] Abdel-Khalik, A. R., and K. M. El-Sheshai, "Information Choice and Utilization in an Experiment on Default Prediction," *Journal of Accounting Research*, (1980), Autumn, pp. 325-342.

[2] Ahmad, S., "Scaling and Generalization in Neural Networks: A Case Study," in *Proceedings of the 1988 Connectionist Models Summer School*, (Eds.) D. Touretzky, G. E. Hinton, T. J. Sejnowski, Palo Alto, CA: Morgan Kaufmann, 1988.

[3] Anderson, J. A., "Neural Models with Cognitive Implications," in *Basic Processes in Reading*, Laberge and Samuels (eds.) , Lawrence Erlbaum, Hillsdale, NJ (1977).

[4] Angulin, D., and C. H. Smith, "Inductive Inference: Theory and Methods," *Computing Surveys*, (1983), 15, 3, pp. 237-269.

[5] Alterman, R. B., R. A. Avery., R. A. Eisenbeis, and J. F. Sinkey, jr., *Application of Classification Techniques in Business, Banking, and Finance*, Greenwich, CT: JAI Press (1981).

[6] Baum, E. B., and D. Haussler, "What Size Net Gives Valid Generalization," *Neural Computation*, 1, (1989), pp. 151-160.

[7] Becker, Sue., and Yann le Cun, "Improving the Convergence of Back-Propagation Learning with Second Order Methods," *Proceedings of the 1988 Connectionist Models Summer School*, (1988), Pittsburgh.

[8] Braun, H., and J. Chandler, "Predicting Stock Market Behavior through Rule Induction: An Application of the Learning-from-examples Approach," *Decision Sciences*, (1987), pp. 415-429.

[9] Breiman, L., J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*, Monterey, CA: Wadsworth (1984).

[10] Carbonell, J. G., and R. S. Michalski, and T. M. Mitchell, "Machine Learning: A Historical and Methodological Analysis," *AI Magazine*, (1983), 4, 3, pp. 69-79.

[11] Carter, C., and J. Cartlett, "Assessing Credit Card Applications Using Machine Learning," *IEEE Expert*, (1987), Fall, pp. 71-79.

[12] Currim, I. S., R. J. Meyer, and N. T. Le, "Disaggregate Tree-Structured Modeling of Consumer Choice Data," *Journal of Marketing Research*, August, (1988), pp. 253-265.

[13] Deprit, E., "Implementing Recurrent Back-Propagation on the Connection Machine," *Neural Networks*, (1989), 2, pp. 295-314.

[14] Dutta, S., and S. Shekhar, "Bond Rating: A Non-conservative Application of Neural Networks," *International Joint Conference on Neural Networks*, Vol. II, (1988), pp. 443-450.

[15] Fahlman, S. E., "Faster-Learning Variations on Back-Propagation: An Empirical Study," *Proceedings of the 1988 Connectionist Models Summer School*, (1988), Pittsburgh.

[16] Finney, D. J., *Probit Analysis*, 3rd. edition, Cambridge University Press (1971).

[17] Fisher, D. H., and K. B. McKusick, "An Empirical Comparison of ID3 and Back-propagation," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, (1989), Detroit, MI, August.

[18] Gentry, James., P. Newbold, and D. Whitford, "Classifying Bankrupt Firms with Funds Flow Components," *Journal of Accounting Research* , (1985), 20, 1, Spring, pp. 146-160.

[19] Gill, Philip E., Walter Murray, and Margaret H. Wright, *Practical Optimization* , Academic Press (1981).

[20] Hinton, G. E., "Learning in Parallel Networks," *BYTE*, (April 1985), pp. 265-273.

[21] Hopfield, J. J., and D. Tank, "Computing with Neural Circuits: A Model," *Science*, (1986), 233, pp. 624-633.

[22] Hornik, K., M. Stinchcombe, and H. White, "Multilayer Feedforward Networks are Universal Approximators," *Neural Networks*, 2, (1989), pp. 359-366.

[23] Koehler, Gary J., and Antal Majthay, "Generalization of Quinlan's Induction Method," *Technical Report, Department of Decision and Information Sciences, The University of Florida, Gainesville, FL.*, August 14 (1988).

[24] Kohonen, T., *Self-Organization and Associative Memory*, Springer-Verlag, Berlin (1984).

[25] Kung, S. Y., and J. N. Hwang, "An Algebraic Projection Analysis for Optimal Hidden Units Size and Learning Rates in Back-Propagation Learning," *Proceedings of the IEEE International Conference on Neural Networks*, (July 1988).

[26] Kung, S. Y., J. Vlontzos, and J. N. Hwang, "VLSI Array Processors for Neural Network Simulation," *Journal of Neural Network Computing*, Spring 1990, pp. 5-20.

[27] Lippmann, R. P., "An Introduction to Computing with Neural Networks," *IEEE ASSP Magazine*, (1987), April, pp. 4-22.

[28] Matheus, C. J., and W. E. Hohensee, "Learning in Artificial Neural Systems," *Computational Intelligence Journal*, (1987), vol. 3, 4.

[29] Messier, W. F. jr., and J. V. Hansen, "Inducing Rules for Expert System Development: An Example Using Default and Bankruptcy Data," *Management Science*, (1988), 34, 12, pp. 1403-1415.

[30] Michaelsen, R. H., "An Expert System for Tax Planning," *Expert Systems*, (1984), October, pp. 149-167.

[31] Michalski, R. S., "A Theory and Methodology of Inductive Learning," *Artificial Intelligence*, (1983), 20, pp. 111-161.

[32] Mingers, J., "Rule Induction with Statistical Data - A Comparison with Multiple Regression," *Journal of Operations Research Society*, (1987), 38, 4, pp. 347-351.

[33] Mooney, R., J. Shavlik, G. Towell, and A. Gove, "An Experimental Comparison of Symbolic and Connectionist Learning Algorithms," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, (1989), Detroit, MI, August.

[34] Orgler, Y. E., "A Credit Scoring Model for Commercial Loans," *Journal of Money, Credit and Banking*, 2, 4, (1970), pp. 435-445.

[35] Parker, D. B., "Optimal Algorithms for Adaptive Networks: Second Order Back-Propagation, Second Order Direct Propagation, and Second Order Hebbian Learning," *Proceedings of the IEEE International Conference on Neural Networks*, (1987), pp. 593-600, San Diego, CA.

[36] Quinlan, J. R., "Induction of Decision Trees," *Machine Learning*, (1986), 1, pp. 81-106.

[37] Rangwala, S. S., and D. A. Dornfeld, "Learning and Optimization of Machining Operations using Computing Abilities of Neural Networks," *IEEE Transactions on Systems, Man, and Cybernetics*, (1989), vol.19, 2, pp. 199-214.

[38] Rendell, L. A., "A General Framework for Induction and a Study of Selective Induction," *Machine Learning*, (1986), 1, pp. 177-226.

[39] Rumelhart, David E., James L. McClelland, and the PDP Research Group, *Parallel Distributed Processing - Explorations in the microstructure of Cognition, Volume I: Foundations* , The MIT Press (1986).

[40] Sejnowski, T. J., and C. M. Rosenberg, "Parallel Networks that Learn to Pronounce English Text," *Complex Systems*, (1987), 1, 1, pp. 145-168.

[41] Shaw, M. J., and J. Gentry, "Inductive Learning for Risk Classification," *IEEE Expert*, (1990), February, pp. 47-53.

[42] Singleton, J. C., and A. J. Surkan, "Modeling the Judgement of Bond Rating Agencies: Artificial Intelligence Applied to Finance," *1990 Midwest Finance Association Meetings, Chicago, IL*, (1990), February.

[43] Walsh, G. R., *Methods of Optimization*, John Wiley (1975).

[44] Watrous, Raymond L. (1987), "Learning Algorithms for Connectionist Networks: Applied Gradient Methods of Nonlinear Optimization," *Proceedings of the IEEE International Conference on Neural Networks*, pp. 619-627, (1987), San Diego, CA.

[45] Weiss, S. M., and I. Kapouleas, "An Empirical Comparison of Pattern Recognition, Neural Nets, and Machine Learning Classification Methods," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, (1989), Detroit, MI, August.

[46] Werbos, P., *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, Ph.D. Thesis, Harvard University (1974).

[47] Zhang, X., M. McKenna, J. P. Mesirov., and D. Waltz, "An Efficient Implementation of the Backpropagation Algorithm on the Connection Machine CM-2," *Technical Report, Thinking Machines Corporation*, (August 29, 1989).

Appendix-A: ID3, and NEWQ

The two inductive learning systems (ID3 and NEWQ) as well as Probit are briefly discussed in this section.

ID3

ID3(Quinlan, 1986) uses information-theoretic measures to generate a decision tree with the classifications at the leaf-nodes.

Let the number of classes in the instance space of interest be C ($C_1, C_2, \ldots, C_c$). At any given node, the expected information that is required to classify the instances at that node as belonging to class C_i or otherwise is given by

$$I(p,n) = -\frac{p}{(p+n)} * log_2[\frac{p}{(p+n)}] - \frac{n}{(p+n)} * log_2[\frac{n}{(n+p)}]$$

where, p is the proportion of the instances at that node that belong to class C_i and n ($= 1 - p$) is the proportion of instances at that node that do not belong to class C_i.

Let the attributes in the instance space be A_j (j = 1..m) where each of the attributes take on V_j values. At a given node let the number of instances that take on the different values for A_j for class C_i be p_i and let $n_i = 1 - p_i$. The expected information required for that branch of the tree with A_j as the root node is

$$E(A_j) = \sum_i \frac{(p_i + n_i)}{(p+n)} * I(p_i, n_i)$$

a weighted average of the proportion of instances that are present at the node of interest. The information that is gained by branching on A_j is

$$gain(A_j) = I(p,n) - E(A_j)$$

The attribute which gains the most information at a node is chosen as the one based on which the branching is performed. The process is used recursively at each node of the tree until all the terminal nodes (leaves) are reached.

NEWQ

NEWQ (Koehler and Majthay, 1988) uses discriminant analysis procedures to produce a decision tree in which a discriminant function (Fisher, 1936) is used at each node for splitting.

Let w be a n-dimensional vector (corresponding to n attributes), and z be a scalar (class value). For w non-zero, (w,z) defines a hyper-plane {x: w'x = z}.

The algorithm begins by generating a hyper-plane (w,z) that strictly separates at least one point from the remaining points in the example set (T). For a 2-class problem, a hyper-plane is used to partition T into {x: $w'x < z, x\epsilon$ T} and {x: $w'x \geq z, x\epsilon$ T}. A new rule is created which augments the old rule set and NEWQ is recursively called with each resulting partition. The reader is referred to Koehler and Majthay (1988) for an elaborate discussion of the NEWQ algorithm.

Appendix-B

Default Classification Data

Abdel-Khalik and El-Sheshai (1980) had previously used this data set to classify a set of firms into those that would default and those that wouldn't default on loan payments. This data set has 32 examples, for training, of which 16 belong to the default class and the other 16 examples belong to the non-default class, and 16 examples for testing, all of which belong to the non-default class. The 18 attributes in the example set are: (1) Net income/total assets, (2) Net income/sales, (3) Total debt/total assets, (4) Cash flow/total debt, (5) long- long-term debt/net worth, (6) Current assets/current liabilities, (7) Quick assets/sales, (8) Quick assets/current liabilities, (9) Working capital/sales, (10) Current year equity/total debt, (11) Sales trend, (12)Earnings trend, (13) Current ratio trend, (14) Working capital/sales trend, (15) Cash flow/total debt trend, (16) Long-term debt/net worth trend, (17) Net income/total assets trend, and (18) Net income/sales trend.

Bankruptcy Prediction Data

This data set is from the *Standard and Poors' Compustat 1981 Industrial Annual Research file of Companies* and *Compustat Industrial files*, and was used to determine the companies that failed during the period 1970-1981. The data was used to predict bankruptcy of firms and is discussed in detail in Gentry, et al.(1985). There are 9 continuous variables in the data set -

ratios of total net flow (TNF), funds from operations (NOFF), working capital which includes inventory, other current assets and liabilities and accounts payable (NWCFF), financial (NFFF), fixed coverage expenses (FCE), capital expenditures (NIFF), dividends (DIV), other assets and liability flows (NOA&LF), and change in cash and marketable securities (CC) with total assets (TA) - with 104 examples and two classes (either bankrupt or not).

Loan Evaluation Data

The loan evaluation data set was used to classify the riskiness involved with firms' abilities to pay back loan. The riskiness is rated from 1 (very risky) to 5 (safe), which form the 5 classes in the study using this data set. There are 15 variables - 12 continuous and 3 nominal - total net flow/total assets, operations, accounts receivable, inv, othca, accounts payable, othcl, otha&l, financial, fcexp, invest, dividend, and three nominal variables guarantee, collateral, and secured. There are 100 examples is this data set with 5 classes.

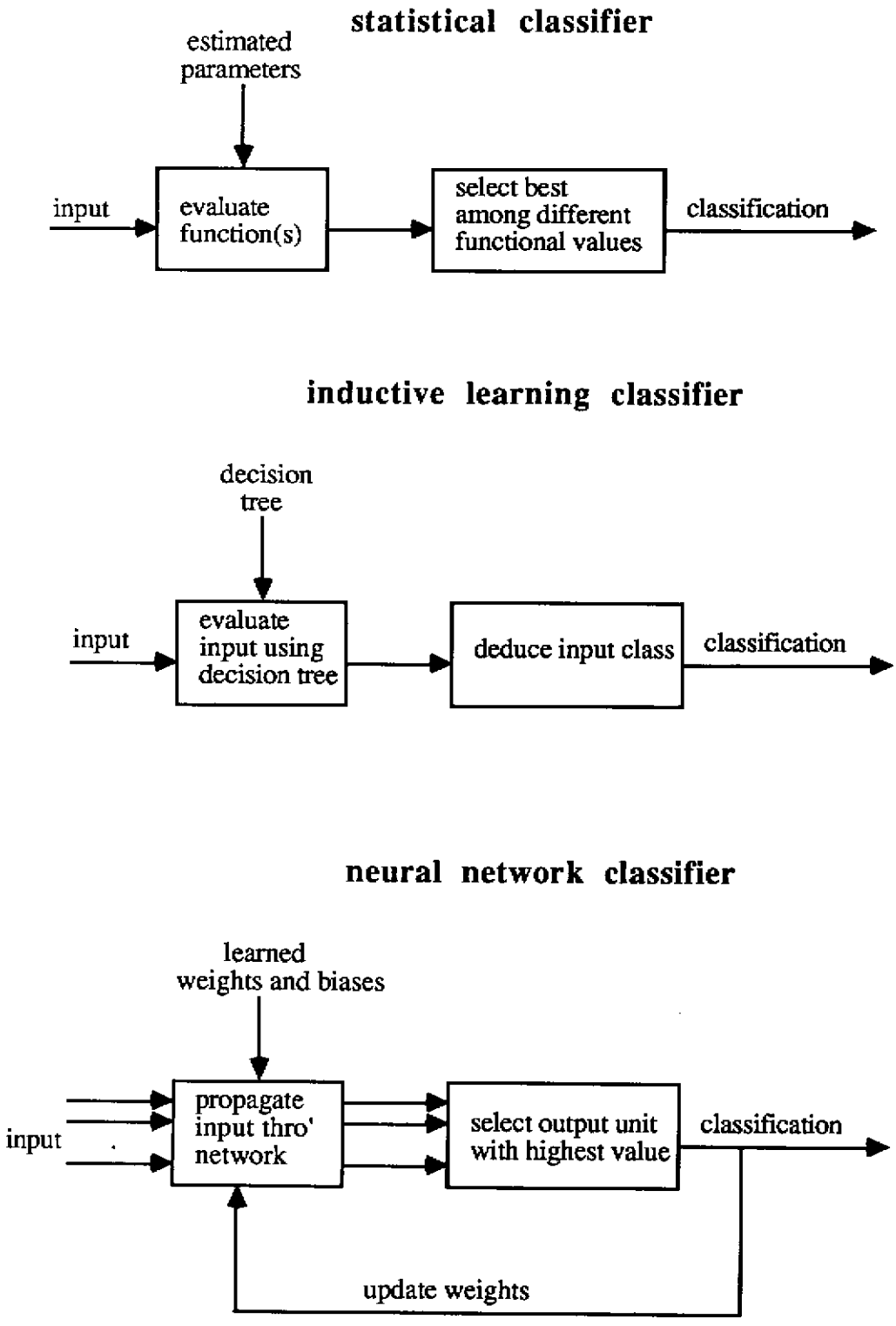

Figure 1: Block diagrams of classifiers

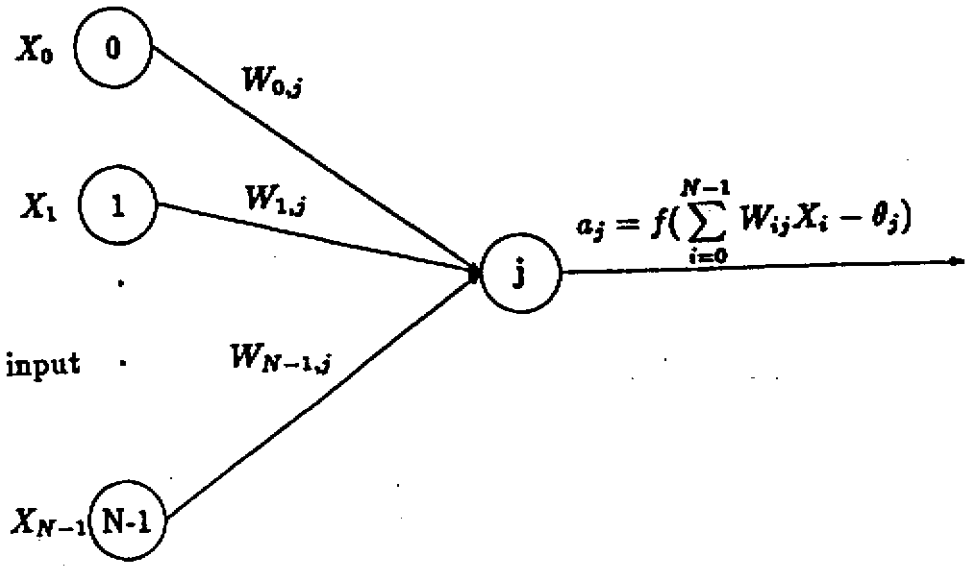

Figure 2: An Artificial Neural Unit, Uj, and Its Activation Function

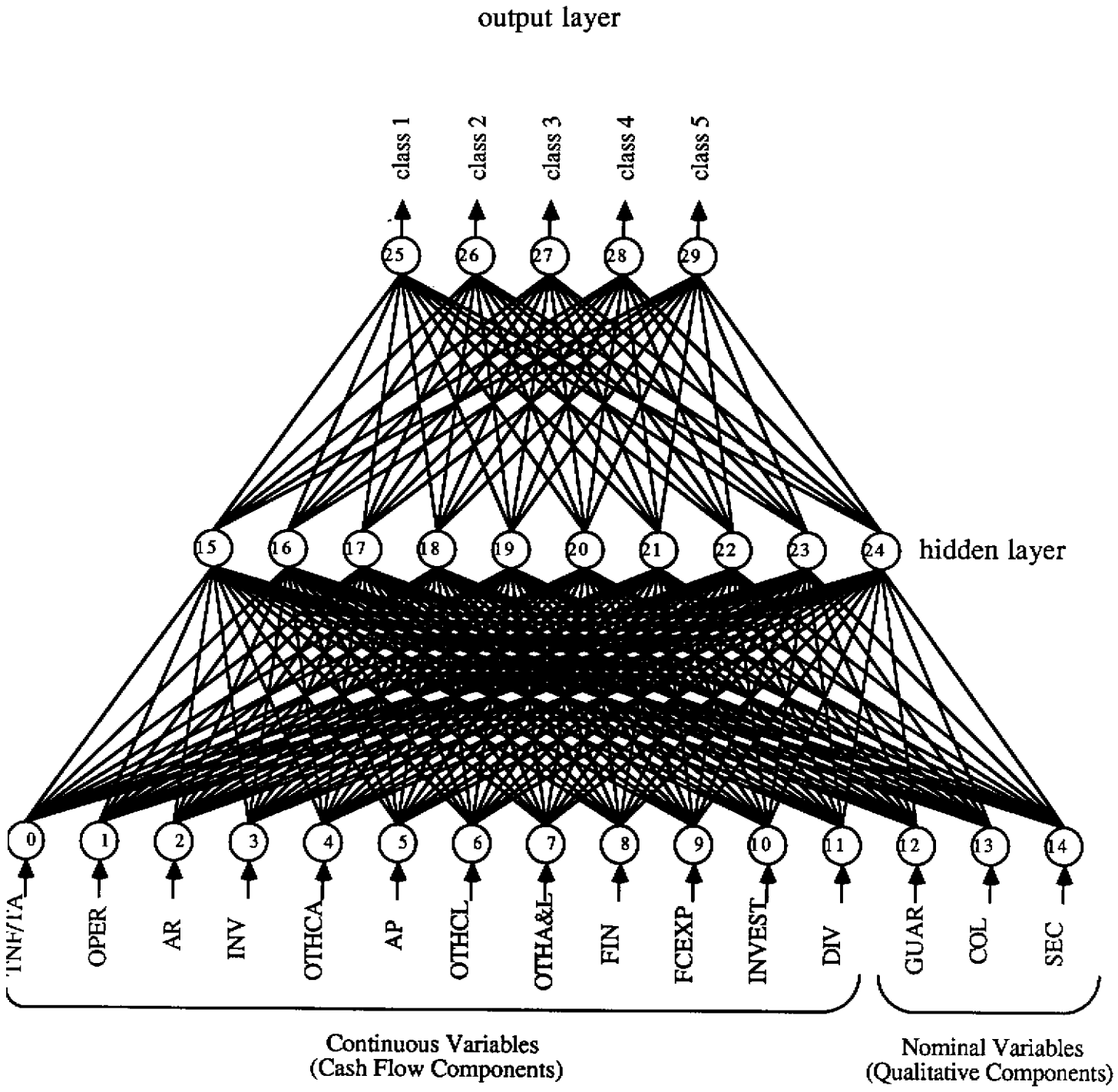

Figure 3: Network for Loan Evaluation Data

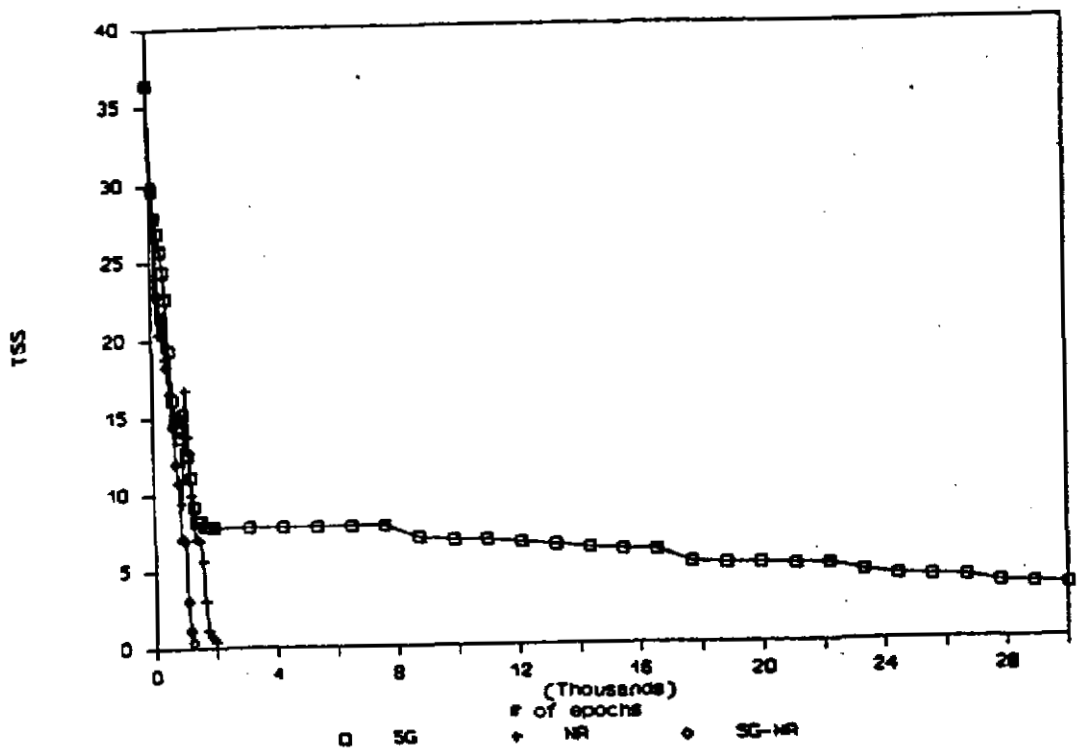

Figure 4: Convergence Rates of SG, NR, and SG-NR for the Example Problem

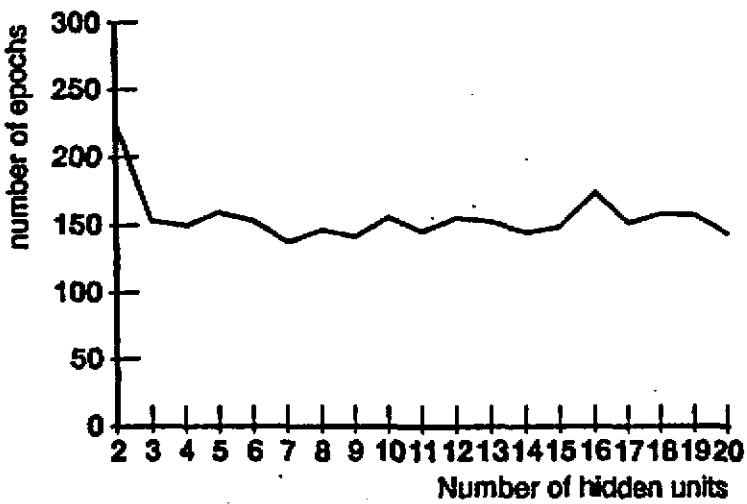

Figure 5: Convergence Speed Dynamics with Varying Number of Hidden Units (Default Classifiaction Data)

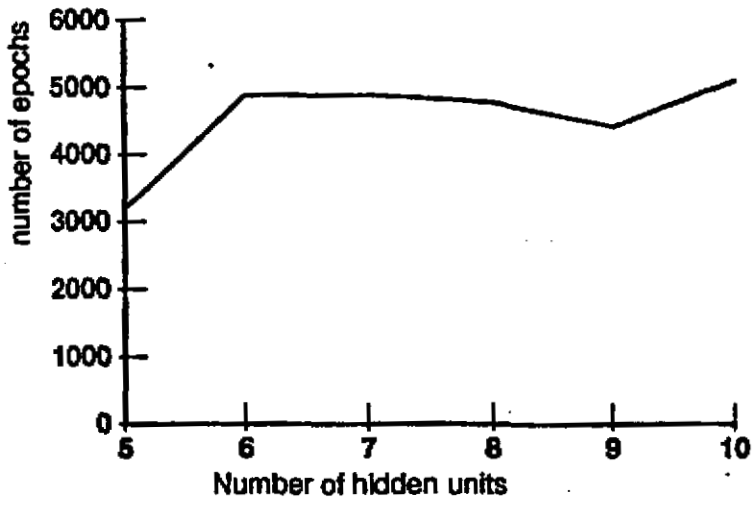

Figure 6: Convergence Speed Dynamics with Varying Number of Hidden Units (Bankruptcy Prediction Data)