Using Feature Construction to Improve the Performance of Neural Networks

Selwyn Piramuthu
Decision and Information Sciences
University of Florida

Harish Ragavan
Beckman Institute for Advance Science and Technology
University of Illinois

Michael J. Shaw
Department of Business Administration
University of Illinois

Bureau of Economic and Business Research
College of Commerce and Business Administration
University of Illinois at Urbana-Champaign
Using Feature Construction to Improve the Performance of Neural Networks

Selwyn Piramuthu
Harish Ragavan
Michael J. Shaw
Using Feature Construction to Improve the Performance of Neural Networks

Selwyn Piramuthu¹
Harish Ragavan²
Michael J. Shaw²,³

March, 1993

Abstract

Recent years have seen the growth in popularity of using neural networks in business decision support because of its capabilities for modeling, estimating, and classifying. Compared to other AI methods for problem solving such as expert systems, neural-network approaches are especially useful for their ability to learn from observation and make adjustments adaptively. However, neural-net learning performed by algorithms such as backpropagation (BP) are known to be slow due to the size of the search space involved and also the iterative manner in which the algorithm works. In this paper, we show that the degree of difficulty in neural-net learning is inherent in the given set of training examples. By identifying a technique for measuring such learning difficulty, we are able to develop a methodology based on feature construction, that helps transform the training data so that both the learning speed and estimation accuracy of neural-net algorithms are improved. We show the efficacy of the method for financial risk classification, a domain characterized by frequent data noise, a lack of functional structures, and high attribute interactions. Moreover, the empirical studies also provide insights into the structural characteristics of neural networks with respect to its training examples and possible mechanisms to improve the learning performance.

¹Decision and Information Sciences, University of Florida
²Beckman Institute for Advanced Science and Technology, University of Illinois
³Department of Business Administration, University of Illinois at Urbana-Champaign
1 Introduction

Recent years have seen the growth in popularity of using neural networks for business decision support due to their excellent performance capabilities for modeling, estimation, and classification. For example, Business Week [1992] described successful implementations of neural networks in a variety of financial applications including market analysis, bond rating and credit evaluation in financial institutions, major corporations and credit rating agencies. Practitioners of management science are interested in applying neural-net methods because of their efficacy for solving complex classification problems and, more significantly, their ability to learn from observations and mistakes. However, neural-net learning algorithms are known to be slow due to the size of the search space involved and also the iterative manner in which the algorithm works.

In this paper, we show that the degree of difficulty in “training” a neural network (i.e., learning) is inherent in the given set of training examples. By developing a technique for measuring this learning difficulty, we are then able to develop a methodology, referred to as feature construction, that helps transform the training data so that both estimation accuracy and the computational times of neural-net algorithms are improved.

Assessing a firm’s financial risk has always been an important decision problem for investors, companies that extend credit, and financial institutions. An incorrect valuation of potential risks can result in serious financial loss. Three aspects of financial risk classification are critical but difficult: the development of a compact model, the use and refinement of the classification model for evaluation, and the identification of relevant financial features. For typical classification problems, values for a set of independent variables are given in a set of observations (i.e., training examples), upon which a model is developed to categorize future observations into appropriate classes. Typical classification problems arise in credit or loan evaluation [Carter and Cartlett, 1987; Orgler, 1970], bond rating [Ang et al., 1975], market survey [Currim, Meyer, and Le, 1988], tax planning [Michaelsen, 1984], and bankruptcy prediction of firms [Hansen and Messier, 1988; Shaw and Gentry, 1990], among other applications.

Neural-net algorithms are beginning to be applied in a wide variety of domains to solve complex problems, including such areas as pattern recognition, category formation, speech understanding, and global optimization [Rumelhart et al., 1986; Sejnowski and Rosenberg, 1987; Hopfield and Tank, 1986]. Most statistical methods applied to business classification applications are limited by assumptions about the distribution of data, independence among the variables, and linearity of the classification model definitions. By contrast, an inherent advantage of back-propagation with neural-nets is that it is affected by these restrictions to a much lesser degree. [Dutta and Shekhar, 1988; Rangwala and Dornfeld, 1989; Collins et al., 1988]. Due to its distribution of knowledge
among neurons, the neural-network method is more tolerant of noise. Moreover, unlike expert systems that use only deductive reasoning, these neural-networks can "learn" new knowledge while solving problems.

Because of these potential advantages, neural-net learning has been increasingly used to solve business classification problems. Of the various connectionist algorithms, back-propagation (BP) is among the most commonly used for classification problems [Rumelhart et al., 1986; Tam and Kiang, 1992].

The thrust of this research is to show that typical business classification problems in management science have their intrinsic structure defined by the training data set and the corresponding search space. Based on this concept, we develop a theoretical measure, referred to as $\Delta$, to characterize this intrinsic structure. This characterization then can be used as a yardstick to guide the improvement of neural-net learning by an induction method called feature construction.

The underlying rationale stems from the fact that neural-net learning is a process to establish a classification model to represent the training data; such a model thus depends strongly on the way the training data are given. Because of the complex interactions among variables and high degree of noise and fluctuations, a majority of data used for classification in business applications are available in representations that are difficult to learn. Transforming the data into a more appropriate representation eases the learning process. In general, training data that are difficult to learn usually demonstrate high dispersion in the search space due to the inability of the low-level measurement attributes to describe the concept concisely. In determining companies' financial risk, for example, it is much more difficult to learn the underlying classifying concept from raw accounting data than from higher-level characterizations such as leverage, liquidity, profitability, growth of sale, and operating cash flows. Given any set of features (attributes) for data representation, it is therefore important to estimate the difficulty of learning the underlying concept(s) using that training data. The learning system should then seek to transform the representations into a space that is easier for learning purposes.

Feature construction builds new representations from the original data, and can be used to reduce the degree of dispersion in the search space within which learning occurs. In this study, we use a feature construction system called FC to construct new features. The new features are used as input to the BP algorithm, to improve its performance. To evaluate the proposed approach, we use a set of boolean data and a real-world risk-classification data set. The resulting performance shows improvement in various performance measures over using just back-propagation. Equally important, we show an approach based on feature construction to transform (and simplify) the search space within which learning takes place, similar to the way principal components are
used to transform the data space in discriminant analysis.

This paper is organized as follows: section 2 evaluates the appropriateness of neural networks for business applications; the concept of measuring learning difficulty is introduced and discussed in section 3; symbolic feature construction and its various characteristics as well as the proposed methodology of integrating symbolic feature construction and back-propagation are briefly discussed in section 4; results using a synthetic boolean classification data and a real-world financial risk-classification data are given in sections 5 and 6 respectively; sections 7 and 8 contain discussion of beneficial effects of the proposed methodology and concludes with a discussion of the results.

2 Neural-Net Learning in Financial Classification

The construction of the classification function $u(x)$ from observations $x$ and the corresponding classification $y$ is a complex and well-researched problem. Traditionally, parametric methods such as multiple discriminant analysis [e.g., Abdel-Khalik and El-Sheshai, 1980], probit [e.g., Finney, 1971], logit, and regression [e.g., Gentry et al., 1985] have been applied. Parametric statistical methods require that the data used to follow a specific distribution (usually Gaussian). In addition, statistical methods have strong restrictions, which could lead to potential problems. While using qualitative variables in probit, when the probit regression lines are not parallel, interpretation of any comparison between them is difficult [Finney, 1971]. Unless their regression coefficients are zero, omitted variables could be a cause for non-zero mean value of error terms in regression analysis which in turn could lead to erroneous results. Multi-collinearity, a major problem when analyzing real-world data, arises due to inter-dependencies among variables. Auto-correlation, due to correlations between residual or error terms of two or more instances, can result in misleading results. Finally, assumption of regression functions to be linear or quadratic might induce additional bias in estimating parameters. The same problem of deriving $u(x)$ from $x = y$ can be viewed as a learning problem, in which a “concept” $u(x)$ is learned from training examples $x = y$.

Neural-networks learn by modifying weights in the links of the network, and are potentially advantageous over statistical methods. Among the characteristics of neural-networks are their inherent parallelism and tolerance to noise, achieved by the distribution of knowledge across the network [Matheus and Hohensee, 1987]. Neural-networks are capable of learning incrementally, thus easing the process of updating knowledge as new instances are obtained. The noise-tolerance feature of neural networks and the ability to represent/learn any function [Hornik et al., 1989] are also very beneficial in business decision-making situations where noise in data is inevitable.
Comparing neural-network methods with other classification methods, a number of prior studies [e.g., Dutta and Shekhar, 1988; Fisher and McKusick, 1989; Mooney, Shavlik, Towell and Gove, 1989; Singleton and Surkan, 1990; Weiss and Kapouleas, 1989] have found that the back-propagation algorithm achieves higher asymptotic accuracy levels and is able to handle noise, albeit requiring a larger training set. Recently, Hansen et al. [1992] compared the performances of a generalized qualitative-response model, neural network, and tree induction, using two problem domains associated with audit decision making and concluded that the former two performed better than the latter and that the results using neural networks showed smaller variance.

In this paper, we use BP as the representative neural-net learning algorithm. BP is naturally amenable to being used for classification since inputs to the BP algorithm are feature values, and the categorization of a given input instance is the corresponding output of the network. Problems of this type are very commonly encountered in business decision making settings. Examples include risk classification, loan evaluation, credit analysis, and financial performance prediction. For these business applications, given data (instances) from previous periods, we are interested in developing models (i.e., learning) to be able to predict future outcomes using just the input feature values.

As stated previously, an inherent problem with the algorithm is that it is very slow to converge in the learning process. Both the learning speed and the accuracy are of primal importance for decision support in business applications. For example, in checking a customer's credit for processing credit-card transactions, the on-line decision support system for authorization needs to be able to respond in 3 to 5 seconds while looking for charges that fall outside the typical credit patterns. Researchers in the area have successfully implemented various modifications for faster convergence of neural networks. This has motivated research [e.g., Fahlman, 1988; Becker and Le Cun, 1988] to alleviate the problem with convergence speed through varied means.

Most common approaches to improving neural-net learning procedures, such as the back-propagation algorithm, use more sophisticated gradient search (e.g., second-order gradient search) techniques instead of the simplistic steepest-descent gradient search process as in the classical back-propagation algorithm. The rationale behind using a second-order gradient search is to be able to take advantage of the inflections in the search space for more efficient search. By focusing on the shape of the search space, the algorithm is able to take appropriate step-lengths in the appropriate direction, thus converging more rapidly towards a solution. Several researchers [Becker and Le Cun, 1988; Fahlman, 1988; Parker, 1987; Waltrous, 1987] have successfully modified the BP algorithm using second-order gradient search methods resulting in improved performance. Another approach that has been widely used is to dynamically configure the network as
learning progresses [e.g., Fahlman, 1988]. This results in the selection of an appropriate (rather than a random) number of hidden units for a given network. Several researchers have utilized Genetic Algorithms for configuring networks used with BP [e.g., Miller, Todd and Hegde, 1989; Montana and Davis, 1989]. Direct modifications to the BP algorithm are just one way to improve its convergence speed. Another means of improving the performance of BP considerably is by taking advantage of the inherent parallelism in the back-propagation algorithm and utilizing highly parallel computers [Hinton, 1985; Deprit, 1989]. In this paper, we present a methodology for improving the learning process in a feed-forward neural network by integrating BP with inductive feature construction.

3 Reducing Learning Difficulty and Its Estimation

The concept learning problem can be defined as deriving the classification “concept” \( \mu(x) \) from the training examples \( x = y \) and the search process is for determining the best description of \( \mu \) that can correctly classify a data case \( x \) with the given attribute values based on the classification underlying the training set \( x = y \). \( \mu(x) \) corresponds to a nonlinear function of \( x \) in neural-net learning.

The concept learning problem can be represented by an instance space composed of the attributes used in the training examples as the axes. For example, Figure 1 represents the instance space of a given concept learning problem, where concepts are represented by membership functions characterizing positive training examples. The circled regions belong to the positive classification with given class memberships. The concept learning process searches through the instance space based on training examples that provide a profile of the concept description to be learned.

When there are multiple regions (peaks) in the instance space, the learning problem is difficult because of the additional search effort needed to cover the disparate regions representing instances belonging to various classes. These types of problems can be characterized as “hard concept learning” [Rendell and Seshu, 1990] for their inherent learning difficulty.

The hard learning problems can also be viewed from the perspective of knowledge representation. Most existing learning techniques, such as neural networks, employ training data with a predetermined set of attributes. In most hard learning problems, however, incorporating the appropriate set of attributes is critical for the success of the learning process and therefore, by itself, is an important decision. In the game of checkers, for example, detailed attributes such as the content of each board position may not be as helpful for learning good strategies as higher-level information, such as piece advantage and mobility. It is therefore reasonable to hypothesize that the learning of checker strategies based on observing the content of board positions is more difficult than the learning.
problem based on training examples from observations described by piece advantage and mobility.

The same phenomenon with respect to the relationship between learning difficulty and the proper representation of the training examples is especially pronounced in the financial risk evaluation domain. In determining companies' credit worthiness, for example, the attributes used in training determine the learning complexity to a great extent, and sometimes even the degree of eventual success of the learning process itself. The credit worthiness of companies would be much more difficult to learn from raw accounting data (e.g., those from the income statements and balance sheets) than from higher-level financial concepts such as liquidity, leverage level, profit growth, and operating cash flow. As successful learning hinges on the proper representation of training examples, two factors are crucial for successful learning. First, there must be a yardstick for measuring the quality, with respect to ease of learning, of the training examples, of a given representation. Second, the learning process should be able to transform the representation used in the training examples, and to seek the most relevant information represented in the training examples.

The purpose of this paper is to describe a methodology which helps achieve these two tasks in neural-net learning. The instance-space paradigm described earlier can help shed light on a possible way to measure learning difficulty. Consider a restricted integer domain with training examples selected on two attributes in the range \([1, 10]\). Figure 2 shows several different types of learning problems in this domain.
Figure 2. Instance Spaces of Learning Problems with Varying Dispersion Numbers of Peaks
From the examples described in Figure 2, at least two major factors should be taken into account in considering learning difficulty: (1) the number of peaks, and (2) their dispersion.

In neural-net learning, these two measurements affect both the network configuration and the convergence speed, as they affect the number of hidden-layer nodes necessary for the learning process as well as the search complexity. Larger numbers of peaks in the instance space imply a greater need to deploy more hidden-layer nodes to account for the various "regions." The dispersion of peaks in the instance space indicates the level of interaction between attributes and thus directly affects the level of search effort required. For instance, Figure 2(c) shows a greater amount of interaction between $x_1$ and $x_2$; therefore, the neural-network requires greater search effort to learn the appropriate connection weights.

For difficult concepts, each projection of the training data produced by conditioning on any attribute value would contain several positive and negative examples, and show high uncertainty about the concept class. Entropy measures this uncertainty—the entropy of a boolean concept $y$ is defined as $H(y) = -(p \log_2 p + n \log_2 n)$ where $p$ and $n$ are the probabilities of finding a positive or negative instance of $y$.

Using this property as a basis for estimating concept difficulty, we can measure the net conditional entropy in the training data, using all the attributes on which the concept depends, i.e., all the relevant attributes. The dispersion $\Delta$ of a concept $y$ is

$$\Delta = \frac{1}{N_{eff}} \sum_{i=1}^{N_{eff}} H(y|x_i)$$

where

- $N_{eff}$ : number of relevant attributes
- $x_i$ : $i^{th}$ relevant attribute
- $H(y|x_i)$ : entropy of $y$ conditioned on $x_i$.

The entropy of $y$ conditional on $x_i$ is defined as

$$H(y|x_i) = -\sum_j \sum_k p(x_i = j)p(y = k|x_i = j)\log_2 p(y = k|x_i = j)$$

over all values $k$ of $y$ and $j$ of $x_i$. $\Delta$ has a value between 0 and 1.

---

1"Relevant attributes" are those attribute $x_i$ whose conditional entropy $H(y|x_i)$ is closer to 0. Having $H(y|x_i)$ close to 1, on the other hand, implies that $x_i$ is not adding any more information about the concept.
Entropy captures the homogeneity of data with respect to data cases of different classes. It provides a yardstick for the degree of uncertainty in the data set: the higher the entropy value, the greater the uncertainty is in the data. Conditional entropy of \( H(y/x_i) \) takes a one-dimensional projection on \( x_i \). For the data corresponding to a hard learning problem, the one-dimensional projection provided by \( H(y/x_i) \) is a mixed spectrum of intertwined positive and negative examples. To estimate the learning difficulty of a given set of training examples, the net uncertainty can be estimated by one-dimensional projections using attributes that are relevant for the learning process, excluding redundant attributes. This set of attributes can be selected (i.e., \( N_{\text{eff}} \)) by estimating the value of \( H(y/x_i) \) corresponding to each attribute and eliminating those attributes whose conditional entropy values are close to 1.

Consider Figure 2; the \( X_1 \) values are sufficient to determine the classification of any example search space depiction in Figure 2(b), whereas both \( X_1 \) and \( X_2 \) values are necessary to determine the class of any example in Figure 2(c). Figure 2(c) clearly illustrates the interaction effects between the axes (\( X_1 \) and \( X_2 \)). This interaction effect necessitates more number of hyperplanes to be able to separate examples belonging to the two classes. The more the number of hyperplanes that are required the harder it becomes for the neural-net to learn the given concepts. This learning difficulty is reflected in their \( \Delta \) values. The single peak data in Figure 2(a) has the lowest \( \Delta \) value of 0.14. In Figure 2(b), \( \Delta(2 \text{ peaks}) = 0.24 \) while \( \Delta(3 \text{ peaks}) = 0.31 \). Figure 2(c) has a more complicated instance space in that interactions between \( X_1 \) and \( X_2 \) should be taken into account in deciding the class membership. Based on that instance space, \( \Delta(2 \text{ peaks}) = 0.29 \) and \( \Delta(3 \text{ peaks}) = 0.43 \).

We used 2-2-1\(^2 \) feed-forward neural-networks and used exhaustive samples from both Figures 2(b) and (c) to train the neural-networks with the back-propagation algorithm. The Figure 2(b) case converged after 395.9 (9.5\(^3 \)) epochs and 12 (1.95) seconds, whereas the Figure 2(c) case did not converge even after 15,000 epochs. From this simple example, we can see that the ease of learning using backpropagation algorithm in a feedforward neural-network is illustrated to be proportional to \( \Delta \) values. This observation leads to the following observation:

**Proposition 1:** \( \Delta \) measures attribute interaction and learning difficulty.

If any single feature splits the positive and negative examples cleanly, such a feature alone is sufficient to determine the concept; no uncertainty would result when the instances are conditioned on such a feature. At the other extreme, all the attributes may need to be simultaneously specified to describe a concept peak. In general, the more

\(^2\)2-2-1 \( \equiv \) 2 input nodes, 2 hidden nodes in a hidden layer and an output node.

\(^3\)Standard deviation values from 10 different backpropagation runs are given in parentheses.
difficult a concept, the higher is its $\Delta$.

4 Feature Construction

4.1 Feature Construction for Reducing Concept Difficulty

Feature construction can be defined in terms of concept learning as follows: *Feature Construction is the process of applying a set of constructive operators* $\{o_1, o_2, \ldots, o_n\}$ *to a set of existing features* $\{f_1, f_2, \ldots, f_m\}$ *resulting in the construction of one or more new features* $\{f'_1, f'_2, \ldots, f'_n\}$ *intended for use in describing the target concept*. A separate learning method (e.g., neural-net learning or similarity-based rule learning) can then make use of the constructed features in attempting to describe the target concept.

Examples of feature construction systems include CITRE [Matheus and Rendell, 1989], FRINGE [Pagallo, 1989], STAGGER [Schlimmer and Fisher, 1986], BACON [Langley et al., 1987], and CINDI [Callan & Utgoff, 1989].

BACON [Langley et al., 1986], a program that discovers relationships among real-valued features of instances in data, uses two operators $(\text{multiply}(\ldots))$ and $(\text{divide}(\ldots))$. This strong bias of restricting the constructive operators allowed, leads to manageable feature construction process, although concept learning would be restricted severely by these chosen operators.

FRINGE [Pagallo, 1989] is a decision-tree based feature construction algorithm. The decision tree is constructed using a similarity-based learning approach. New features are constructed by conjoining pairs of features at the fringe of each of the positive branches. During each iteration, the newly constructed features and the existing features are used as input space for the SBL algorithm. This process is repeated until no new features are constructed. FRINGE alleviates the replication problem by adding a new feature to represent replication thus resulting in succinct encoding of necessary information to describe the concepts more concisely and accurately.

CITRE [Matheus and Rendell, 1989] and DC Fringe [Yang et al., 1991] use a variety of operands such as root (selects the first two features of each positive branch), fringe (similar to FRINGE), root-fringe (combination of both root and fringe), adjacent (selects all adjacent pairs along each branch) and all (all of the above). All of these operands use conjunction as the operator. In DC Fringe, both conjunction as well as disjunction as operators are utilized.

As feature construction proceeds iteratively, the addition of new features to the previous set of features can lead to a large number of features being used as input to the decision tree construction algorithm. Thus, pruning of features is done during each iteration. The most desirable features are kept to be carried over to the next iteration, as well as to form newer features, whereas the least desirable features are discarded. This is
done by the decision tree algorithm (e.g., ID3) through pruning, as well as by the features that were not used in the formation of the decision tree.

Procedure FC (input: Inductive Tree)

Features: = NIL

For every nleaf at depth ≥ 2 in Inductive Tree

If nleaf is a positive leaf then

If (sibling of nleaf is a negative leaf)

And

(nleaf’s parent’s sibling is a positive leaf)

Then Feature : = Disjoint (nleaf)

Else Feature : = Conjoint (nleaf)

Features : = Features + Feature

Return (output: Features)

Detailed steps for constructing Inductive Tree can be found in Quinlan (1986). FC basically resolves the interactions among attributes by conjoining and disjoining features that appear close to the leaf nodes in a decision tree generated by an inductive learning program such as ID3 [Quinlan, 1986]. We use the FC algorithm to construct new feature sets which are easier for learning. Using Δ as an indicator of feature quality, we show that its value typically decreases in the successive feature spaces constructed by algorithms such as FC.

FC constructs features iteratively from decision trees. It forms new features by conjoining as well as disjoining two nodes at the fringe of the tree – the parent and grandparent nodes of positive leaves are conjoined or disjoined to give a new feature. New features are added to the set of original attributes and a new decision tree is constructed using the maximum information-gain criterion [Quinlan, 1986]. This feature selection phase thus chooses from both the newly-constructed features as well as the original attributes for rebuilding the decision tree. The iterative process of tree-building and feature construction continues until no new features are found. Splitting continues to purity, i.e., no pruning [Breiman et al., 1984] is used in this study.

**Proposition 2:** The feature construction process transforms the instance space (of the training examples) and helps decrease the learning difficulty as measured by Δ. Let a given set of training examples be X, and the transformed training examples by feature construction X_{FC}, then Δ(X_{FC}) ≤ Δ(X).
Consider the XOR example in Figure 3(a). This problem requires at least two hyperplanes (straight lines in this space) to be able to separate examples belonging to the two (+, −) classes. The addition of a new feature $X_3$ ($X_3 = X_1 \land X_2$) decreases the learning difficulty by requiring just one hyperplane (abcd in Figure 3b) to be drawn that separates examples belonging to the two classes. Although the addition of a new feature increased the number of effective features used, the resulting space simplified the classification process.

![Figure 3](image)

<table>
<thead>
<tr>
<th>#</th>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_3$</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 3. A new feature (i.e., $X_3$) makes learning easier
Since $\Delta$ measures the difficulty of learning concepts as measured by the dispersion of examples belonging to various classes, the resulting space representing the training examples has a smaller $\Delta$ value. This also follows from Proposition 2.

4.2 Enhancing Neural-Network Learning by Feature Construction

In this study, we combine the process of symbolic feature construction and neural-net learning with back-propagation to form a hybrid system. Inductive feature construction improves the representation of data by providing a compact representation. Back-propagation algorithm has excellent generalization properties. By integrating the two, the beneficial aspects of both can be realised resulting in a better classification system. The data used as input to the BP algorithm are pre-processed appropriately through symbolic feature construction to achieve better performance. More specifically, the complexity of learning the concept, as measured by $\Delta$ defined in the preceding section is reduced through FC for in the attributes used as input to BP, enabling it to learn more effectively [Ragavan and Piramuthu, 1991]. A subset of the original and newly-constructed attributes that have better representations are used as input to the BP algorithm. The new representation has fewer concept regions per class, which makes the search space less complex and possibly reduces the number of hyper-planes needed to separate examples belonging to different classes. The number of hyper-planes required to learn a concept is one of the main determinants of BP convergence speed. When this number is reduced through feature construction, there is a corresponding increase in the convergence speed of BP.

**Proposition 3:** A decrease in the $\Delta$ value of training examples, $X$, is directly proportional to an improvement in the ease of learning from $X$. This should be reflected by the performance of the learning algorithm applied.

This proposition results from the observation that a decrease in the $\Delta$ value results in lesser complex (fewer “peaks”) in the search space. It is easier to learn concepts when the space spanned by the concepts are less complex, since fewer hyperplanes are sufficient to separate examples belonging to different classes than otherwise. Generally, the fewer the number of hyperplanes separating various classes in the spanned space, the more generalizable are the obtained results. The improved generalizability is observed by improved prediction performance of the learned concepts. In the next two sections, we show the effects of the approach with two sample applications.

**Proposition 4:** For a given set of training examples $X$ and its transformed version by feature construction $X^{FC}$, if the Back-propagation procedure is used to train the neural-nets and then test them on hold-out samples for prediction, $X^{FC}$ should help produce better learning performance than $X$ (in terms of the convergence rate and prediction accuracy).
In addition to the mode of search (steepest descent, second-order gradient, conjugate gradient, or other types of gradient search methods) used in the back-propagation algorithm, good features (quality of data) are extremely important. Regardless of the sophistication of gradient search method that is used, an inappropriate set of features can delay or even prevent convergence. When data are the only source of information for searching for good classifications, the characteristics of the instance space must be amenable to yield the expected classifications. Given a fixed representation, the best we can do is to search for a solution in the sub-space covered by the range of values of the known features that are deemed to be important. Consequently, the performance of any learning algorithm is dependent on the quality of the feature set used for representing the data. Hence, selection of the initial set of features plays a crucial role in the learning process.

5 The Effects of Feature Construction on Neural-Net Learning

There are two properties that we would like to stress. First, the characteristics of $\Delta$ in the learning of classifying concepts using neural networks from a set of training examples; second, the impact of feature construction in neural-net learning. As a first step, we use three boolean concepts, defined as $y_1$, $y_2$, and $y_3$ in Disjunctive Normal Form, to illustrate these properties:

$$y_1 = \bar{x}_6 \bar{x}_7 x_8 + \bar{x}_7 x_4 \bar{x}_8 + \bar{x}_3 x_5 x_7$$

$$y_2 = \bar{x}_6 x_1 x_8 + x_8 x_4 \bar{x}_1 + \bar{x}_9 \bar{x}_8 x_1$$

$$y_3 = x_1 \bar{x}_3 \bar{x}_9 + \bar{x}_1 \bar{x}_6 \bar{x}_2 + x_8 \bar{x}_1 x_2$$

Uniformly distributed data were generated for $y_1$, $y_2$ and $y_3$, and used as input to FC. Figure 4 shows the $\Delta$ values using the feature sets selected by FC during the different tree generations, evaluated for all three concepts. (The declining trend of $\Delta$ achieved by feature construction verify Propositions 2 and 3.) The $\Delta$ values drop significantly as new feature sets are used; also, fewer features are used for tree generation. Feature construction is thus used to reduce $\Delta$. The features from each tree are used as input nodes in the BP algorithm. As we hypothesize in Propositions 2 and 3, decreasing the concept’s dispersion in this manner speeds up the convergence of the BP algorithm greatly.

We shall further study the effects of feature sets’ quality on BP performance. A newly-generated feature set is good if it has small $\Delta$ values, relative to the initial feature set. Feature spaces with reduced $\Delta$ values have fewer concept regions, and are thus relatively easier for learning, i.e., for separating the examples belonging to different classes. The
boolean concepts are useful to illustrate the effects of decreasing $\Delta$ on the convergence speed of BP.

![Graph showing the effect of Feature Construction Procedure on Learning Difficulty.](image)

**Figure 4.** The Effect of Feature Construction Procedure on Learning Difficulty.

As the initial weights in the feed-forward neural network were set randomly, we ran the BP algorithm five times for each set of features corresponding to the various trees constructed by FC. The average of five BP runs and their standard deviations are given in Table 1. We use a-b-c, to represent the network configuration in the first column - a, b, and c are the number of input, hidden, and output units respectively. To maintain order in the selection of the number of hidden units, we decided on using half the total number of input and output units as the number of hidden units for all the networks. (This is a rule of thumb suggested in [Rumelhart et al., 1986]). The output layer always has one unit which classifies an example as either positive or negative. The input units were totally connected to the units in the hidden layer, and the units in the hidden layer were totally connected to the units in the output layer.

In Table 1, the decision trees constructed by FC are indicated by $t_{mn}$, for the tree constructed after the $(n-1)^{th}$ iteration for the function $y_m$. The identical entries in
Table 1 for the rows corresponding to the last two trees of each function (e.g., $t_{25}$ and $t_{36}$) are due to the identical final trees that FC produces on convergence. The decision attributes used in the final trees ($t_{15}, t_{26}, t_{36}$) are fewer than the nine in the initial set. This reduces the number of input units, which in turn reduces the hidden units that are necessary. The total number of units used in the network is thus reduced.

Table 1: Results using BP for the boolean concepts.

<table>
<thead>
<tr>
<th>NF</th>
<th>Tree</th>
<th>$\Delta$</th>
<th># of epochs</th>
<th>Time secs.</th>
<th>CUs</th>
<th>NON</th>
</tr>
</thead>
<tbody>
<tr>
<td>9-5-1</td>
<td>$t_{31}$</td>
<td>0.91</td>
<td>107.0 (6.8)</td>
<td>57.6 (3.3)</td>
<td>4815</td>
<td>15</td>
</tr>
<tr>
<td>4-3-1</td>
<td>$t_{12}$</td>
<td>0.87</td>
<td>136.4 (22.2)</td>
<td>4.2 (1.2)</td>
<td>1636.8</td>
<td>8</td>
</tr>
<tr>
<td>4-3-1</td>
<td>$t_{13}$</td>
<td>0.58</td>
<td>95.4 (4.6)</td>
<td>3.2 (0.8)</td>
<td>1144.8</td>
<td>8</td>
</tr>
<tr>
<td>3-2-1</td>
<td>$t_{14}$</td>
<td>0.41</td>
<td>76.4 (8.9)</td>
<td>2.0 (0.0)</td>
<td>458.4</td>
<td>6</td>
</tr>
<tr>
<td>3-2-1</td>
<td>$t_{15}$</td>
<td>0.41</td>
<td>76.4 (8.9)</td>
<td>2.0 (0.0)</td>
<td>458.4</td>
<td>6</td>
</tr>
<tr>
<td>9-5-1</td>
<td>$t_{21}$</td>
<td>0.91</td>
<td>111.2 (6.1)</td>
<td>58.4 (3.5)</td>
<td>5004</td>
<td>15</td>
</tr>
<tr>
<td>6-4-1</td>
<td>$t_{22}$</td>
<td>0.92</td>
<td>123.4 (3.3)</td>
<td>13.6 (0.5)</td>
<td>2961.6</td>
<td>11</td>
</tr>
<tr>
<td>5-4-1</td>
<td>$t_{23}$</td>
<td>0.91</td>
<td>99.0 (2.8)</td>
<td>10.2 (1.0)</td>
<td>2772</td>
<td>10</td>
</tr>
<tr>
<td>5-3-1</td>
<td>$t_{24}$</td>
<td>0.58</td>
<td>62.8 (4.4)</td>
<td>3.2 (0.4)</td>
<td>942</td>
<td>9</td>
</tr>
<tr>
<td>4-3-1</td>
<td>$t_{25}$</td>
<td>0.41</td>
<td>52.8 (3.2)</td>
<td>3.0 (0.0)</td>
<td>633.6</td>
<td>8</td>
</tr>
<tr>
<td>4-3-1</td>
<td>$t_{26}$</td>
<td>0.41</td>
<td>52.8 (3.2)</td>
<td>3.0 (0.0)</td>
<td>633.6</td>
<td>8</td>
</tr>
<tr>
<td>9-5-1</td>
<td>$t_{31}$</td>
<td>0.93</td>
<td>115.4 (18.1)</td>
<td>61.8 (9.4)</td>
<td>3193</td>
<td>15</td>
</tr>
<tr>
<td>6-4-1</td>
<td>$t_{32}$</td>
<td>0.94</td>
<td>115.8 (4.7)</td>
<td>8.6 (0.5)</td>
<td>2779.2</td>
<td>11</td>
</tr>
<tr>
<td>6-4-1</td>
<td>$t_{33}$</td>
<td>0.81</td>
<td>77.6 (2.7)</td>
<td>5.4 (0.5)</td>
<td>1862.4</td>
<td>11</td>
</tr>
<tr>
<td>5-3-1</td>
<td>$t_{34}$</td>
<td>0.58</td>
<td>66.0 (5.8)</td>
<td>4.6 (0.5)</td>
<td>990</td>
<td>9</td>
</tr>
<tr>
<td>4-3-1</td>
<td>$t_{35}$</td>
<td>0.41</td>
<td>57.2 (4.6)</td>
<td>2.4 (0.5)</td>
<td>686.4</td>
<td>8</td>
</tr>
<tr>
<td>4-3-1</td>
<td>$t_{36}$</td>
<td>0.41</td>
<td>57.2 (4.6)</td>
<td>2.4 (0.5)</td>
<td>686.4</td>
<td>8</td>
</tr>
</tbody>
</table>

Legend:

NF: Network Configuration (#input-#hidden-#output)
$\Delta$: Learning Difficulty
CUs: Number of Connections Updated
NON: Number of Neurons Used in the Network

Except for a few cases, the standard deviation (shown in parentheses) of each value is low compared to its mean value. The standard deviation values do not seem to have any specific pattern with respect to the number of units used in the neural network.
A closer look at the first two performance criteria in Table 1 is instructive. The number of epochs required for convergence shows a slight initial increase in some cases, but then reduces considerably as better representations are constructed. The number of epochs taken by the final set of features \((t_{15}, t_{26}, t_{36})\) to converge decreases to about half the value corresponding to the original attributes \((t_{11}, t_{21}, t_{31})\), for all three examples. The time taken for BP to converge drops precipitously for all three concepts as the tree generation proceeds, before finally levelling off. The BP convergence time for the final features is less than an order of magnitude compared to those using the original set of attributes.

This trend of improved performance with decreasing concept difficulty is also clear from the decreasing number of connection updates \((CU = \# \text{ of epochs} \times \text{total number of weights in the network})\) in Table 1. The time taken by the different networks does not correspond strictly to their CUs probably because of the arithmetic operations (differing numbers of zero values in the various connections). The reduction in convergence time is substantial due to significant drop in the number of connection updates as newer feature sets are generated. Because of serial processing, the time taken per epoch depends to a large extent on the total number of units that are used in the network. This is not the case if parallel processors (e.g., a connection machine) are used for the units. Parallel updating of activations in a layer in the network reduces the time taken per epoch proportional to the number of units in the layer.

In summary, the impacts of feature construction on neural-net learning, as shown in this example, are the following:

- the reduction of learning difficulty.
- the reduction of the network size necessary for classification.
- the reduction of learning time.

Furthermore, feature construction should also help improve the predictive accuracy of the learned model, as stated in Proposition 4. This property can be illustrated better by financial risk evaluation applications discussed in the following section.

6 Applications in Financial Risk Classification

As it is important for companies, investors, and financial institutions to assess firms' financial health or riskiness, numerous empirical models have been developed that use annual financial information to distinguish between firms that are healthy and the ones that are risky (for example, Abdel-Khalik and El-Sheshai, 1980). Although the bankruptcy literature is extensive, research interest continues in the development of a theoretical
foundation that would capture the many dimensions of financial distress and failure. Likewise, numerous lenders and investors want to improve their ability to explain, interpret, and predict bankruptcy.

This type of financial risk analysis presents a challenge to the development of appropriate classification models because of the lack of linear relationships among attributes, the inherent level of noise in the training data, and the high degree of interactions among attributes. Gentry et al. [1985] use cash flow information to provide unique insights into the prediction of bankruptcy, bond rating, and loan risk ratings. We use bankruptcy data in this study; half of the companies went bankrupt in a given period while the other half were financially healthy during the same time period. The cash flow model given in Appendix A was used for the first 11 attributes. Besides funds flow components, we also included additional financial attributes such as the ratio of the total cash flow/total asset, accumulated depreciation/fixed asset, and change of sales volume. These attributes are represented as \( x_1, \ldots, x_{14} \) in Figure 5. Each of the 182 companies falls in one of two classes: the positive examples (class 1) represent non-failed companies, and the negative examples (class 0) represent failed companies. We used holdout samples (about 10% of the total) to evaluate the performance of the learned weights in the neural networks. A typical feed-forward neural network that is used in this study (corresponding to the features shown in Figure 5) is shown in Appendix B.

The empirical results using the proposed algorithm for financial risk classification data confirms our previous results with boolean data. Table 2 summarizes the (average) results of our experiments using the financial risk classification data. The values given in Table 2 are all averaged over five different runs of the BP algorithm. \( X^{F_1}, X^{F_2}, X^{F_3}, X^{F_4}, \) and \( X^{F_5} \) correspond to five different sets of constructed features generated using the original data set. The average performance over all these trees (\( X^{F_1}, X^{F_2}, X^{F_3}, X^{F_4}, \) and \( X^{F_5} \)) are given by \( X^{F_{\text{average}}} \) in Table 2.

Unlike in Table 1 where the progress of the feature construction process was shown in sequence, Table 2 contains only the final acts of features that were generated using the feature construction algorithm. We used 10% of the sample for testing purposes and these were sampled randomly from the whole data set. Thus, \( X^{F_1} \) through \( X^{F_5} \) were generated using different samples (of 90% from the original data set) and were the final sets of features during individual runs (after convergence) of the feature construction algorithm. Table 2 shows that the neural network learned from \( X^{\text{original}} \) is less desirable than that learned from any of the transformed training data sets, \( X^{F_1} \) to \( X^{F_5} \), in (1) network size, (2) time to converge, and (3) prediction accuracy.

For learning from the same set of training examples, smaller neural networks generally are considered more favorable than larger ones because small networks are more efficient
in both learning and problem-solving stages. Using the new features generated by FC helps reduce the necessary dimension of the instance space, resulting in smaller neural networks to represent the transformed training examples. This size reduction in the classification model can best be illustrated by the decision tree size reduction achieved by feature construction as illustrated in Figure 6.

\[
\begin{align*}
    f_1 &= ((30 \leq x_1 \leq 49) \land \neg((x_1 \leq 31) \land (x_{13} \leq 38))). \\
    f_2 &= ((x_6 \leq 24) \lor (x_{10} \geq 29)) \land \\
           \neg(((x_7 \geq 22) \land \neg(19 \leq x_9 \leq 49)) \lor [(x_7 \leq 21) \land (x_{10} \leq 25)]). \\
    f_3 &= ((x_1 \leq 32) \land \neg(22 \leq x_4 \leq 49)) \lor ((x_4 \leq 20) \lor \neg(26 \leq x_{13} \leq 49)). \\
    f_4 &= (x_1 \leq 32) \land (x_{13} \leq 35). \\
    f_5 &= (x_{13} \leq 32) \lor (24 \leq x_{10} \leq 49). \\
    f_6 &= ((x_9 \leq 26) \land \neg(25 \leq x_8 \leq 49)) \lor ((x_9 \geq 27) \land (32 \leq x_{12} \leq 59)). \\
    f_7 &= \neg(25 \leq x_5 \leq 49) \land (x_{10} \leq 28) \land (x_1 \geq 24)) \lor \\
           ((25 \leq x_5 \leq 49) \land \{(x_{10} \leq 28) \land (x_1 \geq 24)\} \lor (x_7 \leq 24)). \\
    f_8 &= (23 \leq x_1 \leq 49) \land (22 \leq x_5 \leq 49) \land \{(\neg(29 \leq x_{12} \leq 59) \lor \\
           (31 \leq x_1 \leq 49)) \lor \{(36 \leq x_8 \leq 49) \land (x_7 \geq 26)\} \lor \\
           \{(x_{11} \leq 21) \land (36 \leq x_8 \leq 49)\})). \\
    f_9 &= (x_8 \leq 5). \\
    f_{10} &= (23 \leq x_4 \leq 49) \land (10 \leq x_{12} \leq 36).
\end{align*}
\]

Figure 5. New Features Constructed by FC for $X^{F_1}$.

Table 2 also shows that neural networks learned from the transformed data (i.e., $X^{F_1}$ to $X^{F_5}$) are better predictive models than the ones learned from the original training examples $X^{\text{original}}$. This superiority is demonstrated by the smaller gaps between the accuracy of classifying training examples (Training %) versus the (prediction) accuracy of classifying holdout examples (Testing %).
Table 2: Results using BP for the Financial Risk Evaluation Data.

<table>
<thead>
<tr>
<th></th>
<th>$X_{F_1}$</th>
<th>$X_{F_2}$</th>
<th>$X_{F_3}$</th>
<th>$X_{F_4}$</th>
<th>$X_{F_{average}}$</th>
<th>$X_{original}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Size</td>
<td>10.6-1</td>
<td>10.6-1</td>
<td>10.6-1</td>
<td>11.6-1</td>
<td>11.6-1</td>
<td>14.8-1</td>
</tr>
<tr>
<td># of epochs</td>
<td>138.6 (31.41)</td>
<td>122.4 (18.55)</td>
<td>69.8 (8.47)</td>
<td>52.6 (7.47)</td>
<td>126.2 (11.65)</td>
<td>1056.4 (277.12)</td>
</tr>
<tr>
<td>Time (seconds)</td>
<td>46.2 (8.45)</td>
<td>30 (1.78)</td>
<td>26 (5.02)</td>
<td>15.8 (3.12)</td>
<td>33.1 (3.56)</td>
<td>482.25 (105.81)</td>
</tr>
<tr>
<td>Connection Updates</td>
<td>2356.2</td>
<td>2080.8</td>
<td>1186.6</td>
<td>894.2</td>
<td>2154.1</td>
<td>21297.2</td>
</tr>
<tr>
<td>Training%</td>
<td>84.48 (1.03)</td>
<td>87.55 (1.36)</td>
<td>87.00 (0.34)</td>
<td>81.19 (0.95)</td>
<td>90.91 (0.63)</td>
<td>76.11 (3.77)</td>
</tr>
<tr>
<td>Testing%</td>
<td>8.17 (1.36)</td>
<td>81.11 (1.11)</td>
<td>83.33 (0.00)</td>
<td>84.44 (2.83)</td>
<td>83.59 (1.11)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure 6. Classification Model Reduction Achieved by Feature Construction ((a2) and (b2) are the Decision Trees Using New Constructed Features)

As an indication of improvements on the classification process, the classification accuracy for the data used for the networks using constructed features ($X^{F_1}, X^{F_2}, X^{F_3}, X^{F_4},$ and $X^{F_5}$) are slightly lower than using back-propagation with the original data, and the testing accuracy performances of the neural networks are higher.
In other words, when feature-construction is used, the neural network with the new features generalized more on the data, resulting in reduced classification accuracy. However, this generalization improves the classification and helps improve prediction accuracy. This improvement in predictive performance is achieved by making the neural-net more generalized and less specific to the training examples. Otherwise, the resulting neural-net model with learned connections would be so specific to the training examples that brittleness could arise when new and heretofore unseen examples are presented to the network.

These observations on the performance improvements of neural networks achieved by feature construction can be stated by the following proposition.

**Proposition 5:** The improved neural-net learning performance achieved by feature construction is due to the fact that feature construction helps transform the instance space for neural-net learning into an instance space where the class-membership function has fewer peaks and feature interactions. As a result, fewer hyperplanes are required to learn concepts in this space.

Direct evidence of the better behaved instance space is the reduced $\Delta$ values, as discussed in Propositions 2, 3, and 4. In addition, the empirical study with financial classification applications show that feature construction has also enhanced the additional dimension regarding the performance of neural networks, as measured by the accuracy of prediction of novel examples. This is achieved by aiding neural networks to generalize learned knowledge—in terms of weights in the network—to a level such that it is not too much on the over- or under-generalization side. Although these neural networks classify fewer training cases correctly, the prediction accuracy on the testing data is improved. This convergence of classification (on training data) and prediction accuracies (on testing data) is desirable since it implies that the learned knowledge is less specific to the training data but more generally applicable to other data from the domain of interest. As newer features are constructed, the dispersion of data in the instance space decreases, which in turn increases the ease of learning concepts using the resulting search space. Although learning in a feed-forward network using back-propagation algorithm occurs by a process of search through weight-space in the network, the ease of learning even through weight-space is enhanced by a reduction in dispersion in the instance space.

Neural networks require fewer epochs to learn a concept if its dispersion is decreased by using good features. By constructing new features, we reduce the number of relevant attributes that are needed to define the concepts, and also increase the average information content at each of the constructed input units. This is achieved by compiling the “interaction” effects of the attributes in disjunctive concept terms into features. Using feature construction, the performance of the back-propagation algorithm is thus improved.
in three ways:

1. By reducing the total number of units in the network, the number of activation updates required per epoch is reduced.

2. By increasing the average information content of each feature that is used as input to BP, the number of epochs required for convergence is reduced.

3. By improving the ease of search through the solution-space, appropriate generalizations are achieved by the network, thus leading to improved prediction accuracy.

Hence, the performance of the back-propagation algorithm is improved both in terms of the time taken per epoch (leading to a decrease in the overall time taken), as well as the number of epochs, which translates to reduced connection updates. The learned weights in the network are also generalized such that the prediction accuracy (using the testing examples) is increased.

In this study, for comparable classification results, the time taken by the BP algorithm to converge using the features corresponding to $X^{F_1}$, $X^{F_2}$, $X^{F_3}$, $X^{F_4}$, and $X^{F_5}$ are close to an order of magnitude less than that with the original set of attributes ($X^{original}$). The number of epochs (and therefore the CUs) using the trees with constructed features are also about an order of magnitude less than those compared to that using the original set of attributes. The prediction accuracy increased by about 9%, on an average.

The financial data set that we used in this study certainly is replete with noise as well as the available information itself being prone to incompleteness (such as an incomplete set of attributes as compared to those that are required to be able to classify/predict any data from the domain under consideration). In spite of all these constraints, one should be able to efficiently obtain information from available data so as to compensate for the inadequacies of the available data. Our study has shown that the hybrid approach incorporating feature construction and back-propagation does better even in these noisy conditions (using noisy real-world data), in improving the speed of convergence of the back-propagation algorithm as well as improving the prediction accuracies involved.
7 Discussion

Analyses using financial risk data as well as artificially generated data support the propositions that are given in sections 3 and 4. These can be seen from tables 1 and 2. Table 1 shows that the Δ value of the data set decreases as the process of construction of newer features proceeds. Also, the Connection Update values decrease in most cases as newer features are constructed. These support propositions 1, 2 and 3. The convergence rate of back-propagation algorithm, as measured by the number of epochs required to converge as well as the time taken to converge, also decreases in most cases as newer features are constructed.

We have shown, using synthetic boolean as well as real-world risk-classification data sets, a systematic performance improvement in feed-forward neural networks using the proposed methodology. As a result of feature construction, the dimensionality of the representation-space was reduced, which enabled the data to be represented in a compact format. In addition to compact representation, the process of feature construction also resulted in producing a set of features with greater information content than the initial feature set, as attested by the Δ values. The complexity of the feature space (the number of peaks in the search space) was also reduced through feature construction, thus requiring a reduced number of hyperplanes to separate examples belonging to various categories. The reduced number of features in the feature-set decreased the number of connection updates that were required by the feed-forward neural network before converging to the pre-specified tss value. The number of connection updates are proportional to the number of epochs taken before converging. Thus, fewer number of input nodes to the neural net means a reduction in the time taken per epoch as well as the number of epochs before convergence; it also results in the reduction in the number of peaks in the search space, which enhanced the performance of the back-propagation algorithm in being able to separate examples belonging to different classes using fewer hyperplanes.

The improved information content in the new set of features resulted in improved generalizations and thus improved prediction accuracy results. The overall impact of our methodology is seen from the improved speed of convergence of the feedforward neural network as well as the improved performance of the neural network in terms of prediction accuracy.

Neural networks, on the other hand, help achieve prediction accuracy that would not be possible by using the feature construction algorithm alone. Furthermore, neural networks are good at incremental learning (i.e., the situation where learning is continuously being carried out as new training examples are observed) while feature construction by itself cannot learn incrementally. Our methodology thus nicely creates a synergy between feature construction and neural-networks that improves upon both approaches.
8 Conclusion

The Back-Propagation algorithm is being successfully used in commercial applications, such as credit risk rating of companies. In a commercial credit risk rating situation, for example, performance factors such as prediction accuracy and the learning speed of the algorithm is critical. We have shown a means of getting closer to the goal of achieving better predictive accuracy and faster learning using a feed-forward neural network by automating the input feature selection process. Feature construction can be used to automatically generate better feature sets, as measured by their $\Delta$ values, which are used as input to the BP algorithm. The proposed methodology also eliminates the least important attributes from the training data, thus facilitating efficient use of computing resources by focusing on only those attributes important for a given classification problem.

Given a data set, using feature construction, the ratio of the number of features to the number of examples in the input to the back-propagation algorithm is reduced, which renders learning using back-propagation more statistically valid. By using a set of attributes with reduced $\Delta$, along with other means of increasing the convergence speed such as second-order gradient methods, the convergence speed of the BP algorithm can be significantly improved. The different means of improving the performance of BP can be used to complement one another in achieving a better overall performance. In this paper, we have definitively established the relationship between $\Delta$ and the complexity of learning a neural network from a set of training data.

Advantages of neural networks such as good performance in high feature interaction domains [Fisher and McKusick, 1989] are combined with advantages (e.g., attribute criticality identification, decision structure identification, and knowledge interpretability) of decision-tree induction by our integrated method. Incorporating feature construction into the BP algorithm also provides a technique for introducing domain knowledge in neural nets, where knowledge gets compiled into the constructed features. In other words, our method combines the accuracy and adaptability of neural networks with the knowledge interpretability of feature construction, as illustrated by an application to financial risk assessment.
Appendix A
The Set of Attributes Used in the Analysis

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Explanation (Abbreviation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1$</td>
<td>net operating flow / total cash flow</td>
</tr>
<tr>
<td>$X_2$</td>
<td>net investment flow / total cash flow</td>
</tr>
<tr>
<td>$X_3$</td>
<td>dividends / total cash flow</td>
</tr>
<tr>
<td>$X_4$</td>
<td>fixed coverage expenditures / total cash flow</td>
</tr>
<tr>
<td>$X_5$</td>
<td>changes in receivables / total cash flow</td>
</tr>
<tr>
<td>$X_6$</td>
<td>change in inventories / total cash flow</td>
</tr>
<tr>
<td>$X_7$</td>
<td>change in other current assets / total cash flow</td>
</tr>
<tr>
<td>$X_8$</td>
<td>change in payables / total cash flow</td>
</tr>
<tr>
<td>$X_9$</td>
<td>change in other current liabilities / total cash flow</td>
</tr>
<tr>
<td>$X_{10}$</td>
<td>change in net financial / total cash flow</td>
</tr>
<tr>
<td>$X_{11}$</td>
<td>change in net other assets and liability / total cash flow</td>
</tr>
<tr>
<td>$X_{12}$</td>
<td>total cash flow / total asset</td>
</tr>
<tr>
<td>$X_{13}$</td>
<td>accumulated depreciation / fixed assets</td>
</tr>
<tr>
<td>$X_{14}$</td>
<td>sales trend</td>
</tr>
</tbody>
</table>
Appendix B
The Configuration of the Neural Nets

bankrupt/not bankrupt

\[ f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9, f_{10} \]
References


