The Utility of Feature Construction for Back-Propagation*

Harish Ragavan Selwyn Piramuthu
ragavan@cs.niuc.edu selwyn@dante.cs.uiuc.edu
Beckman Institute, 405 N. Mathews Avenue
University of Illinois at Urbana-Champaign, Urbctna, IL 61801, U.S.A.

Abstract

The ease of learning concepts from examples in empirical machine learning depends on the attributes used for describing the training data. We show that decision-tree based feature construction can be used to improve the performance of back-propagation (BP), an artificial neural network algorithm, both in terms of the convergence speed and the number of epochs taken by the BP algorithm to converge. We use disjunctive concepts to illustrate feature construction, and describe a measure of feature quality and concept difficulty. We show that a reduction in the difficulty of the concepts to be learned by constructing better representations increases the performance of BP considerably.

1 Introduction

Recent progress in *artificial neural networks* (ANNs) and their use in disparate domains have spurred the interests of researchers in studying various means of improving them. ANNs 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* a/., 1986; Anderson, 1977; Sejnowski and Rosenberg, 1987; Hopfield and Tank, 1986]. They have also been tried with fairly good results in financial [Duttaand Shekhar, 1988] and in manufacturing applications [Rangwala and Dornfeld, 1989] among other areas.

Back-propagation (BP) is one of the most widely used neural network algorithms due to its powerful problem solving capabilities. There have been numerous studies by researchers who are interested in the dynamics of the BP algorithm, and the network (multi-layered perceptrons) on which back-propagation is based. Although BP has varied advantages, such as being able to represent/learn any function [Hornik *et* a/., 1989], an inherent problem with the algorithm is that it is very slow to converge (learn). Researchers in the area have successfully implemented modifications to enable fastei convergence of the neural networks using disparate means, ranging from modifications to the BP algorithm itself to appropriate implementations of the algorithm in VLSI components. In this paper, we present a novel technique for improving the learning process in a feed-forward neural network by pre-processing the input (training) data using an automated methodology.

A difficult concept shows a high degree of *dispersion* in the input representation space due to the inability of the low-level measurement attributes to describe the concepts concisely and accurately. Given any feature set for data representation, the *concept dispersion* measure A [Ragavan and Rendell, 1991] estimates the difficulty of learning any concept using that feature set, and thus the quality of the feature set. We use *feature construction* [Pagallo, 1989; Matheus and Rendell, 1989; Drastal *et* a/., 1989] to develop new features from the original set of attributes, to decrease the concept's dispersion in the constructed feature space. The constructed feature sets are then used as input to the BP algorithm. New features are constructed using CITRE [Matheus, 1989] and used to improve the performance of the BP algorithm considerably.

2 Reducing Concept Dispersion by Feature Construction

Difficult concepts exhibit numerous "peaks" or regions in instance space [Rendell and Seshu, 1990] if the attributes used for describing data are inappropriate. These concepts require changes in representation, for example through feature construction. Good feature construction can reduce the difficulty of such concepts, by providing a more compact representation for the training data.

For difficult concepts, each subset of the training data formed by conditioning on a value of any attribute would contain a large number of both positive and negative examples, and consequently show high uncertainty about the concept class. Entropy[1] measures this uncertainty—to measure concept difficulty, we estimate the net conditional entropy in the training data, using all the at-

*This research was supported in part by National Science Foundation grant no. IRI-88-22031. We would like to thank Christopher Matheus and Larry Rendell for providing us a copy of CITRE.

[1] 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 prior probabilities of finding a positive or negative instance of y.

tributes (assumed independent) on which the concept depends.

We define dispersion Δ of a concept y as

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

where

N_{eff} : number of *effective* attributes (those relevant to the concept)
x_i : i^{th} effective attribute
$H(y/x_i)$: entropy of y conditioned on x_i.

The entropy of y conditional on x_i is defined as

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

over all values j of X_i Δ has a value between 0 and 1.0.

In general, the more difficult the concept, the higher its dispersion. At the other extreme, 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. Δ captures the difficulty of a concept for learning using any given feature set. It can therefore be used estimate feature set quality.

We use feature construction, specifically using CITRE [Matheus, 1989], to create new feature sets with lower Δ values. Using Δ as an indicator of feature quality, we show that Δ decreases in the successive feature spaces constructed by CITRE. More importantly, BP performance increases as Δ decreases.

The FRINGE [Pagallo, 1989] module of CITRE constructs features iteratively from decision trees. It forms new features by conjoining two nodes at the fringe of the tree—the parent and grandparent nodes of positive leaves are conjoined 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* a/., 1984] is used. The reader is referred to [Matheus, 1989] for a more detailed discussion on CITRE.

We use three different disjunctive boolean concepts to illustrate our experiments. The three functions are:

$$y_1 = \bar{x}_6\bar{x}_7x_8 + \bar{x}_7x_4\bar{x}_8 + \bar{x}_3x_5x_7$$
$$y_2 = \bar{x}_6x_1x_8 + x_8x_4\bar{x}_1 + \bar{x}_9\bar{x}_8x_1$$
$$y_3 = x_1\bar{x}_8\bar{x}_9 + \bar{x}_1\bar{x}_6\bar{x}_2 + x_8\bar{x}_1x_2$$

Three different data sets were generated for the functions y_1, $y_2$and y_3, and used as input to the FRINGE feature construction module in CITRE. The newly constructed features and the trees that were generated for the first function (y_1) are shown in Fig.I.

The Δ values of the feature sets selected by CITRE during the different tree generations are evaluated. Results for all three disjunctive concepts are shown in Fig.2.

```
Tree generation 1:
 att3 att4 att5 att6 att7 att8.
 New Features:
  11    and(equal(att3,false),equal(att5,true))
  12    and(equal(att4,true),equal(att8,ialse))
  13    and(equal(att8,true),equal(att6,lalse))
Tree generation 2:
 11 12 13 att7.
 New Features:
  14    and(equal(att7,true),equal(11,true))
  15    and(equal(12,true),equal(att7,false))
  16    and(equal(13,true),equal(12,false))
Tree generation 3:
 14 15 13 att7.
 New Features:
  17    and(equal(att7,false),equal(f3,true))
  18    and(equal(15,true),equal(14,1alse))
Tree generation 4:
 14 15 17.
 New Features:
  19    and(equal(f7,true),equal(f5,false))
Tree generation 5:
 14 15 17.
```

Figure 1: Features constructed by CITRE for y_1.

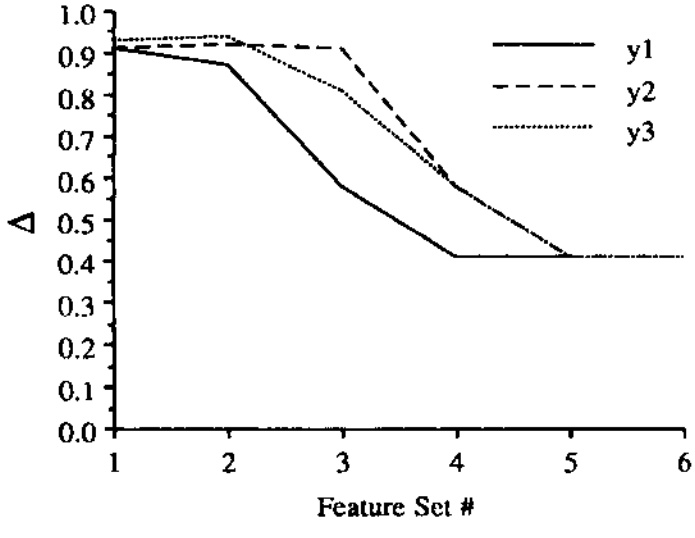

Figure 2: Quality of the Features constructed by CITRE.

As can be seen from this figure, the Δ values drop significantly as new feature sets are used. We use the features from each tree as input to the BP algorithm. As we show in the next section, decreasing the concept's dispersion in the condensed feature sets in this manner speeds up the convergence of the BP algorithm greatly.

3 Learning with Back-Propagation

A number of previous studies [Dutta and Shekar, 1988; Fisher and McKusick, 1989; Mooney *et* a/., 1989; Weiss and Kapouleas, 1989] have compared BP with other classification methods including statistical tree induction, with results favoring BP in terms of classification accuracy. One of the major drawbacks of BP is that it is very slow. Several researchers [Fahlman, 1988; Becker and Le Cun, 1988] have worked on this problem trying to increase the convergence speed of the BP algorithm by disparate means. There are four primary means of increasing the convergence speed of BP: (1) by appropriately pre-processing the data used as input to the algorithm, (2) by improving the BP algorithm itself, (3) by hard-wiring the algorithm using VLSI circuits, and (4) by utilizing the inherent parallelism in the BP algorithm through implementations in parallel machines.

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. Kung, Vlontos, and Hwang [1990] describe a VLSI architecture for implementing BP using a programmable systolic array. Hinton [1985] and Deprit [1989] describe the use of parallel processing computers to implement the BP algorithm with each unit in the network assigned to a processor.

In this study, no attempt is made to improve the BP algorithm itself as in previous studies. Instead, the data used as input to the BP algorithm is pre-processed (see [Piramuthu, 1990]). More specifically the Δ of the concept is reduced in the attributes used as input to BP, to enable it to learn more effectively. As discussed in section 2, feature construction can be used to reduce the Δ of concepts. Using feature construction, a subset of the initial and newly constructed attributes that are deemed to be better for representation are used as input to the BP algorithm. The new representation has fewer concept regions per class. This makes the search space less complex, which in turn increases the convergence speed of BP.

4 Using Good Feature Sets for BP

The criterion we use to categorize a newly generated feature set as "good" is that it should have small Δ values, relative to the initial feature set. Feature spaces with reduced Δ values have fewer concept regions, and are thus relatively easier for learning, i.e., for separating the examples belonging to different classes. The disjunctive concepts of Section 2 are used to study the effect of decrease in Δ on the convergence speed of BP.

As the initial weights in the network were set randomly, we ran the BP algorithm 5 times for each set of features corresponding to the various trees constructed by CITRE. The average of 5 BP runs and their standard deviations are given in Table 1. The number of units in the network[2] is also shown in the table. The number of hidden units is roughly half the total number of input and output units for all the networks. The output layer always has 1 unit which classifies an example as either positive or negative.

Table 1: Results using BP for all three concepts. Standard deviations are shown in parenthesis.

Network	Tree	# of epochs	Time secs.	CUs
9+5+1	t_{11}	107.0 (6.8)	57.6 (3.3)	1605
4+3+1	t_{12}	136.4 (22.2)	4.2 (1.2)	1091
4+3+1	t_{13}	95.4 (4.6)	3.2 (0.8)	763
3+2+1	t_{14}	76.4 (8.9)	2.0 (0.0)	458
3+2+1	t_{15}	76.4 (8.9)	2.0 (0.0)	458
9+5+1	t_{21}	111.2 (6.1)	58.4 (3.5)	1668
6+4+1	t_{22}	123.4 (3.3)	13.6 (0.5)	1357
7+4+1	t_{23}	99.0 (2.8)	10.2 (1.0)	1188
5+3+1	t_{24}	62.8 (4.4)	3.2 (0.4)	565
4+3+1	t_{25}	52.8 (3.2)	3.0 (0.0)	422
4+3+1	t_{26}	52.8 (3.2)	3.0 (0.0)	422
9+5+1	t_{31}	115.4 (18.1)	61.8 (9.4)	1731
6+4+1	t_{32}	115.8 (4.7)	8.6 (0.5)	1274
6+4+1	t_{33}	77.6 (2.7)	5.4 (0.5)	854
5+3+1	t_{34}	66.0 (5.8)	4.6 (0.5)	594
4+3+1	t_{35}	57.2 (4.6)	2.4 (0.5)	458
4+3+1	t_{36}	57.2 (4.6)	2.4 (0.5)	458

In Table 1, the decision trees constructed by CITRE 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_{26}) are due to the identical final trees that CITRE produces on convergence.

The decision attributes used in the final trees (t_{15},t_{26},t_{36}) are fewer than those in the initial set (9). This reduces the number of input units, in turn reducing the hidden units that are necessary. Thus the total number of units used in the network is reduced. Except for a few cases, the standard deviations for each of the resulting values are low compared to their respective mean values. It can also be seen that the standard deviation values do not have any specific pattern with respect to the number of units used in the neural network.

We now take a closer look at the first two performance criteria listed in Table 1. The number of epochs and time taken to converge by BP for the three concepts as a function of constructed feature sets (tree generations) are shown in graphically in Figs. 3 and 4 respectively.

As Fig.3 shows, the epochs required for convergence shows a slight initial increase in some cases, but then reduces considerably as better representations are con-

[2] a+b+c — a, b, and c are the number of input, hidden, and output units respectively.

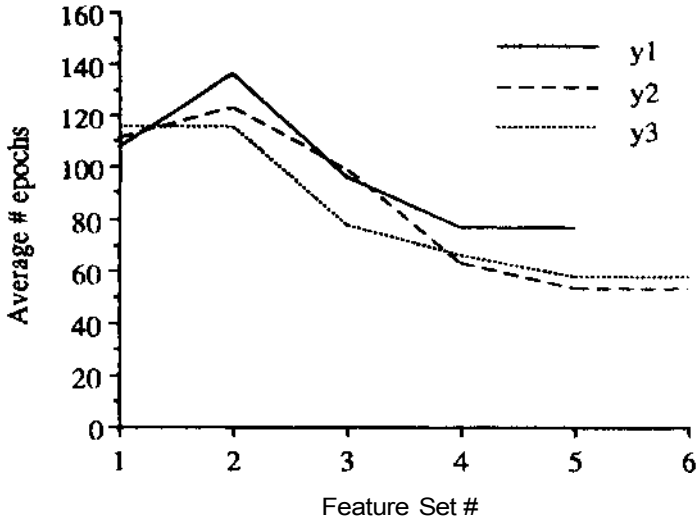

Figure 3: BP Epochs in the new Feature Spaces.

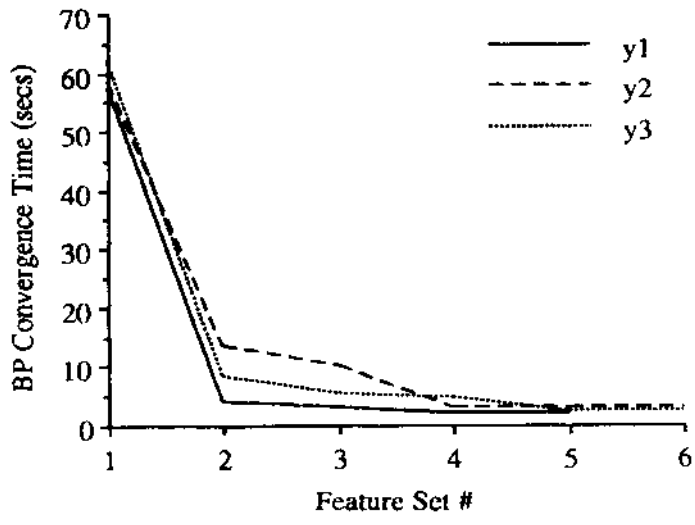

Figure 4: BP Convergence Time improvement.

structed. The number of epochs taken by the final set of features (t_{15}, t_{26}, t_{30}) to converge decreases to about half the value corresponding to the original attributes (t_{11}, t_{21}, t_{31}), for all three examples.

In Fig.4, the time taken for BP to converge, for all three examples, drops precipitously as the tree generation proceeds, before finally levelling off. The time taken for the BP algorithm to converge using the final set of attributes is less than an order of magnitude compared to using the initial attributes.

This trend of improved performance with decreasing concept difficulty is also clear from the reducing number of connection updates (CUs = # epochs x total number of units in the network) in Table 1. 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 would not be the case if parallel processors (e.g., Connection Machine) are used for the units.

Neural networks require fewer epochs to learn a concept if its dispersion is decreased by using good features. By constructing new features, we reduce not only the number of effective attributes that are needed to define the concepts, but also increase the average information content at each of the constructed input units. This is achieved by considering the *interaction effects* of the attributes in disjunctive concept terms in addition to the *main effects* of the individual disjunctive terms, through feature construction.

5 Discussion

The Back Propagation algorithm is being successfully used in commercial applications, such as credit risk rating of companies. There have also been developments in the area of creating expert systems using neural networks. In a commercial credit risk rating situation, for example, the learning speed of the BP algorithm is critical for the firm to be able to make quicker decisions for it to remain competitive. We have shown a means of getting closer to the goal of achieving 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 Δ 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 directing attention to only those attributes important for a given classification problem.

Advantanges of neural networks such as good performance in high feature interaction domains [Indhurkya and Weiss, 1990] are combined with advantages of decision-tree based induction by this method. Incorporating a front-end like CITRE to the BP algorithm also provides a facile technique for introducing domain knowledge in neural nets. Knowledge gets compiled into the constructed features.

It should also be noted that the classification accuracy was a full 100 percent for feature sets corresponding to each of the trees (t_{11}, t_{12}, t_{13}, t_{14}, t_{15}, t_{21}, t_{22}, t_{23}, t_{24}, t_{26}, t_{31}, t_{32}, t_{33}, t_{34}, t_{35}, t_{36}), in spite of the reduction in the number of attributes used as input after feature construction. By using a set of attributes with reduced Δ_1 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. Performance with noisy data remains to be investigated.

References

[Becker and Le Cun, 1988] S.Becker and Y.Le Cun. Improving the convergence of back-propagation learning with second-order methods. *Proceedings of the 1988 Connectionist Models Summer School,* 1988.

[Breiman *et al,* 1984] L. Breiman, J. Friedman, R. Olshen and C. Stone. *Classification and Regression Trees.* Belmont, CA, Wadsworth, 1984.

[Deprit, 1989] E. Deprit. Implementing recurrent back-propagation on the connection machine. *Neural Networks,* 2:295-314, 1989.

[Drastal *et al,* 1989] G. Drastal, G. Czako, and S. Raatz. Learning in an abstraction space: A form of constructive induction. *Proceedings of the Eleventh IJCAI,* 1989.

[Dutta and Shekar, 1988] S. Dutta and S. Shekhar. Bond Rating: A non-conservative application of neural-networks. *International Joint Conference on Neural Networks,* 443-450, 1988.

[Fahlman, 1988] S. E. Fahlman. Faster-learning variations on back-propagation: An empirical study. *Proceedings of the 1988 Connectionist Models Summer School,* 1988.

[Fisher and McKusick, 1989] D. H. Fisher and K. B. McKusick. An empirical comparison of ID3 and back-propagation. *Proceedings of the Eleventh IJCAI,* 1989.

[Hinton, 1985] G. E. Hinton. Learning in parallel networks. *BYTE,* 265-273, April 1985.

[Hopfield and Tank, 1986] J. J. Hopfield and D. Tank. Computing with neural circuits: A model. *Science,* 233:624-633, 1986.

[Hornik *et al.,* 1989] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks,* 2: 359-366, 1989.

[Indurkhya and Weiss, 1990] N. Indurkhya and S. M. Weiss. Iterative Rule Induction Procedures. *Laboratory for Computer Science Research Report LCSR-TR-145,* Rutgers University, New Brunswick, New Jersey, 1990.

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

[Kung *et al,* 1990] S. Y. Kung, J. Vlontzos and J. N. Hwang. VLSI array processors for neural network simulation. *Journal of Neural Network Computing,* 5-20, 1990.

[Matheus, 1989] C. J. Matheus. Feature Construction: An Analytic Framework and An Application to Decision Trees. *Ph.D. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign,* 1989.

[Matheus and Rendell, 1989] C. J. Matheus and L. Rendell. Constructive Induction in Decision Trees. *Proceedings of the Eleventh IJCAI,* pp 645-650, 1989.

[Mooney *et al,* 1989] R. Mooney, J. Shavlik, G. Towell and A. Gove. An Experimental Comparison of symbolic and connectionist learning algorithms. *Proceedings of the Eleventh IJCAI,* 1989.

[Pagallo, 1989] G. Pagallo. Learning DNF by decision trees. *Proceedings of the Eleventh IJCAI,* 639-644, 1989.

[Parker, 1987] D. B. Parker. 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,* 593-600, 1987.

[Piramuthu, 1990] S. Piramuthu. Feature construction for Back-propagation. *Proceedings of the International Workshop on Parallel Problem Solving from Nature,* Springer-Verlag, 1990.

[Quinlan, 1986] J. R. Quinlan. Induction of Decision Trees. *Machine Learning,* Vol 1, No 1, pp 81-106,1986.

[Ragavan and Rendell, 1991] H. Ragavan and L. Rendell. Estimating the utility of feature construction in empirical learning. *Working paper,* University of Illinois, 1991.

[Rangwala and Dornfeld, 1989] S. S. Rangwala and D. A. Dornfeld. Learning and optimization of machine operations using computing abilities of neural networks. *IEEE Transactions on Systems, Man, and Cybernetics,* 19, 2:199-214, 1989.

[Rendell and Seshu, 1989] L. Rendell and R. Seshu. Learning hard concepts through constructive induction: Framework and rationale. *Computational Intelligence,* Volume 6, No. 4, pp 247-270, Nov. 1990.

[Rumelhart *et al,* 1986] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group. *Parallel Distributed Processing - Explorations in the Micro structure of Cognition, Volume I: Foundations.* The MIT Press, 1986.

[Sejnowski and Rosenberg, 1987] T. J. Sejnowski and C. M. Rosenberg. Parallel Networks that learn to pronounce english text. *Complex Systems,* 1, 1:145-168, 1987.

[Waltrous, 1987] R. L. Waltrous. Learning algorithms for connectionist networks: Applied gradient methods of nonlinear optimization. *Proceedings of the IEEE International Conference on Neural Networks,* 619-627, 1987.

[Weiss and Kapouleas, 1989] S.M.Weiss and Kapouleas I. An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. *Proceedings of the Eleventh IJCAI,* 1989.