2007-09-05

Improving Neural Network Classification Training

Michael Edwin Rimer
Brigham Young University - Provo

IMPROVING NEURAL NETWORK CLASSIFICATION TRAINING

by

Michael E. Rimer

A dissertation submitted to the faculty of

Brigham Young University

In partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

Brigham Young University

December 2007

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a dissertation submitted by

Michael E. Rimer

This dissertation has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Tony Martinez, Chair
Date	Michael Goodrich
Date	Dan Olsen
Date	Bryan Morse
Date	Kent Seamons

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the dissertation of Michael E. Rimer in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date	Tony Martinez Chair, Graduate Committee

Accepted for the Department

Date	Parris Egbert Graduate Coordinator

Accepted for the College

Date	Thomas W. Sederberg Associate Dean, College of Physical and Mathematical Sciences

ABSTRACT

IMPROVING NEURAL NETWORK CLASSIFICATION TRAINING

Michael E. Rimer

Department of Computer Science

Doctor of Philosophy

The following work presents a new set of general methods for improving neural network accuracy on classification tasks, grouped under the label of classification-based methods. The central theme of these approaches is to provide problem representations and error functions that more directly improve classification accuracy than conventional learning and error functions.

The CB1 algorithm attempts to maximize classification accuracy by selectively backpropagating error only on misclassified training patterns. CB2 incorporates a sliding error threshold to the CB1 algorithm, interpolating between the behavior of CB1 and standard error backpropagation as training progresses in order to avoid prematurely saturated network weights. CB3 learns a confidence threshold for each combination of training pattern and output class. This models an error function based on the

performance of the network as it trains in order to avoid local overfit and premature weight saturation. PL1 is a point-wise local binning algorithm used to calibrate a learning model to output more accurate posterior probabilities. This algorithm is used to improve the reliability of classification-based networks while retaining their higher degree of classification accuracy.

These approaches are demonstrated to be robust to a variety of learning parameter settings and have better classification accuracy than standard approaches on a variety of applications, such as OCR and speech recognition.

ACKNOWLEDGMENTS

I would like to thank my wife Olga for her emotional, spiritual and physical support and encouragement in producing this work.

I am grateful to my advisor, Dr. Tony Martinez, for his guidance and focus in producing quality research. I thank him and other faculty for astute and helpful comments on the different phases of this work.

I thank all my colleagues and friends with whom I discussed and considered these ideas in much detail, and who freely shared related knowledge to further my studies.

Most deeply, I thank my Heavenly Father for bestowing His creative spark, strength, and inner light, which makes all learning and understanding possible.

Table of Contents

Chapter 1

Introduction

The study of machine learning is largely concerned with improving the automation of decision making processes, ranging from function approximation (regression tasks) to classification or concept learning (selecting from a set of possible choices). Examples of classification are performing medical diagnoses, optical character recognition, speech recognition, and document content identification.

The artificial neural network (ANN) was inspired by biological models of neurological systems and is an established machine learning model with robust learning properties and simple deployment. ANNs are often used as a "black box" that receives data observations as input and outputs decisions based on these observations. This work focuses on studying and improving the theoretical and practical use of artificial neural networks in classification problem domains.

1.1 Issues in learning

In order for a learning model to output educated decisions, it must first be trained. There are various issues involved in training that affect a model's subsequent recognition performance. To train a learner (whether it be a human or a machine), real data observations from the problem domain must be acquired. In many problem domains, it is infeasible to acquire data for, or even model, the complete domain. Hence, *training* consists of encountering a subset of instances picked in some fashion from the entire

problem domain. A training set, T, consists of a finite number, N, of observations. An observation, $\boldsymbol{X}$, may be defined on a vector of k attributes (or features),

$$\boldsymbol{X} = (X_1, X_2, \ldots, X_k),$$

and is given an associated target value, y, indicating the correct decision:

$$\text{Training Set} = \{(\boldsymbol{X}_n, y_n)\ n = 1, 2, \ldots, N\}.$$

In supervised learning, the model's task is to accurately predict values for y given instances of $\boldsymbol{X}$. However, the main goal in learning is not for the model merely to memorize the training data, but rather to *generalize* what the limited training data teach in order to make correct decisions on future data encountered across the entire problem population.

A learner must apply some inductive bias to forming a hypothesis that correctly classifies the training data. However, if the learner *overfits*, or makes false assumptions on the general nature of the problem based on idiosyncrasies in the training set, this can become a detriment to generalization. ANNs, like many other learning algorithms that can form complex hypotheses, are prone to overfitting. There is an inherent tradeoff between fitting the training data perfectly and generalizing accurately over the entire population. Much work has gone into studying how to achieve this balance, with varying degrees of success, but intrinsically this remains an open problem.

1.2 Neural network training issues

The iterative error backpropagation algorithm commonly used to train ANNs is adept at approximating continuous functions of high degree. This learning algorithm can also be used in classification. However, the mathematical representations of a classification task have different properties than that of function approximation, and there are certain distinctions in the training process of each. This leads to important distinctions in the type of learning approach best suited to either task, which relate to avoiding overfit and achieving high generalization.

To generalize well, error backpropagation must use a proper error minimization, or objective, function. A common error function is minimizing *sum-squared-error* (SSE). The validity of using SSE as an objective function relies on the assumption that pattern outputs are offset by inherent Gaussian noise, being normally distributed about a cluster mean. For function approximation of an arbitrary signal, this presumption often holds. However, this assumption is invalid for classification problems where the target vectors are class codings (i.e., arbitrary nominal or boolean values representing designated class labels). In this case, it is better to maximize *cross-entropy* (CE) in order to discriminate among choices. However, these objective functions provide mechanisms that do not explicitly reflect the goal of classification learning (i.e., achieving high recognition rates on unseen data).

1.3 Contribution of this work

The contribution of this work is to improve on the conventional approach of training ANNs by minimizing a global error function. Error functions like SSE and CE, maintaining routine statistical properties and mathematical continuity, are replaced with non-continuous error functions that seek to more directly train ANNs to generalize well on classification problems. These methods are more analogous to how other learning models, like decision trees, perform concept-based learning, while still retaining the simplicity, speed and robustness of ANNs.

This work presents several methods for learning classification tasks that are superior to existing learning algorithms. These classification-based (CB) methods are implemented in various ways, but the general approach is to define an objective function that seeks to directly minimize classification error instead of attempting to approximate a function represented by transformed class labels (i.e., 0/1 targets). This work is a collection of these methods, which have either been published previously or submitted for publication in various refereed journals or conferences. Following is a summary of each chapter paper, followed by the publication reference.

Chapter 2 presents the notion of classification-based objective functions. It discusses properties of backpropagation learning as related to classification performance in detail. *CB1*, the first incarnation of a classification-based objective function, is presented. CB1 does not backpropagate an error signal through the network on correctly classified patterns. CB1 is shown to discourage premature weight saturation and improve

generalization. It performs successfully on speech recognition tasks, a large OCR data set and several benchmark problems selected from the UC Irvine Machine Learning Repository. [Rimer, M. & Martinez, T. (2006). *Classification-based Objective Functions*. *Machine Learning*, vol. 63, no. 2, pp. 183-205.][1]

Chapter 3 illustrates how CB1, also called *lazy training* since it skips training on correctly-classified patterns, may be applied to a multi-tier speech model to improve recognition accuracy over training with CE. [Rimer, M., Martinez, T. & Wilson, D. (2002). *Improving Speech Recognition Learning through Lazy Training*, Proceedings of the IEEE International Joint Conference on Neural Networks IJCNN'02, pp. 2568-2573.]

Chapter 4 presents *CB2*, or *softprop*, a combination of SSE and CB1. It performs error minimization analogously to the "softmax" exploration policy used in Q-learning that combines greedy exploitation with conservative exploration in an optimization search. This exploration policy tends to be effective in complex problem spaces that have many local minima. Implementing this methodology in ANN training is shown to achieve higher test accuracy and more robust solutions than either SSE or CB1. [Rimer, M. & Martinez, T. (2004). *Softprop: Softmax Neural Network Backpropagation Learning*, Proceedings of the IEEE International Joint Conference on Neural Networks IJCNN'04, pp. 979-984.]

[1] This chapter is an extension of work previously published in [Rimer, M. & Martinez, T., IJCNN 2001; Rimer, M., Master's Thesis, Brigham Young University, 2002].

Chapter 5 presents CB3, a classification-based algorithm which performs adaptive local modifications to the error function during training. CB3 is designed to address the concern that ANN overfit is not only a global, but more particularly a local phenomenon, for which adaptive localized learning can be beneficial. Based on the network's success in learning individual training patterns, CB3 adjusts these patterns' intended target values during training by learning individual confidence margins. CB3 exhibits significantly improved generalization over conventional training, CB1, and CB2. [Rimer, M. & Martinez, T. (2006). *CB3: An Adaptive Error Function for Backpropagation Training*, Neural Processing Letters, vol. 24, no. 1, pp. 81-92.]

Chapter 6 performs an in-depth comparative analysis of SSE, CE, CB1, CB2, and CB3, measuring their robustness to initial conditions, learning parameters, and convergence properties. CB3 is shown to be superior with respect to all of these. [Rimer, M. & Martinez, T. (2006). *Analysis of Classification-based Error Functions*, submitted to Machine Learning.]

Chapter 7 presents PL1, a novel method of calibrating a learning model. Calibration refers to how accurate a learning model is in predicting posterior probabilities. Utility theory dictates that, in order for a general learning model to perform optimally, it must output well-calibrated probabilities. In other words, model calibration is important since a learning model does not operate in isolation, but rather outputs confidence values which must subsequently be acted on by an operator to make a final decision. The efficacy of using PL1 to calibrate ANNs in order to reduce probability estimation error is

demonstrated. [Rimer, M., Peterson, A. & Martinez, T. (2007). *Improving Posteriors with Point-wise Local Binning*, submitted to Neural Processing Letters.]

Chapter 8 presents results of applying PL1 to CB algorithms. The above implementations of CB learning are designed to output correct classifications as much as possible, but in doing so, neglect to output accurate probability estimates of these decisions. The PL1 algorithm is used as a post-processing step to calibrate CB1-3 to output more accurate posterior probabilities. In particular, CB3 is shown to operate with similar precision to CE when both are calibrated, while CB3 retains its higher degree of generalization. [Rimer, M. & Martinez, T. (2007). *Calibrating Classification-based networks to improve posteriors*, submitted to Neural Processing Letters.]

Chapter 9 presents an alternate stochastic model for learning classification tasks. Four heuristics are presented for selectively choosing which patterns are presented to the network during training. The primary purpose of this work at the time of publication was to illustrate how functionally redundant training patterns may dynamically be culled from the training set to dramatically increase learning speed without degrading accuracy. [Rimer, M., Andersen, T. & Martinez, T. (2001). *Speed Training: Improving Learning Speed for Large Data Sets*, IJCNN'2001: INNS-IEEE International Joint Conference on Neural Networks, Washington, D.C., pp. 2662-2666.]

The following papers were developed concurrently with the above chapters. Though not directly related to classification-based learning, they provide similar studies in the optimization of ANN training and are included here as appendices for completeness.

Appendix A is a study of the optimal network size for ANNs operating alone in contrast to networks included in an ensemble. It is shown that the optimal size for networks operating in an ensemble is smaller than for a network operating alone on the tested data sets. [Andersen, T., Rimer M. & Martinez, T. (2001). *Optimal Artificial Neural Network Architecture Selection for Bagging*, IJCNN'2001: INNS-IEEE International Joint Conference on Neural Networks, Washington, D.C., pp. 790-795.]

Appendix B presents *oracle learning*, in which we study how a small neural network may be trained to mimic the performance of a much larger network. Having a small network is a practical consideration when developing a recognition system that must have a small memory footprint. It is demonstrated that an oracle-trained network can exhibit better performance than a network of equivalent size trained directly on the training data. [Menke, J., Peterson, A., Rimer, M. & Martinez, T. (2002). *Network simplification through oracle learning*. Proceedings of the IEEE International Joint Conference on Neural Networks IJCNN'02, pp. 2482-2487.]

Chapter 2

Classification-based Objective Functions

MICHAEL RIMER mrimer@axon.cs.byu.edu

TONY MARTINEZ martinez@cs.byu.edu

Computer Science Department, Brigham Young University, Provo, UT 84602, USA

Phone: (801) 422-6464 **Fax: (801) 422-0169**

Abstract. Backpropagation, similar to most learning algorithms that can form complex decision surfaces, is prone to overfitting. This work presents classification-based objective functions, an approach to training artificial neural networks on classification problems. Classification-based learning attempts to guide the network directly to correct pattern classification rather than using common error minimization heuristics, such as sum-squared error (SSE) and cross-entropy (CE), which do not explicitly minimize classification error. CB1 is presented here as a novel objective function for learning classification problems. It seeks to directly minimize classification error by backpropagating error only on misclassified patterns from culprit output nodes. CB1 discourages weight saturation and overfitting and achieves higher accuracy on classification problems than optimizing SSE or CE. Experiments on a large OCR data set have shown CB1 to significantly increase generalization accuracy over SSE or CE optimization, from 97.86% and 98.10%, respectively, to 99.11%. Comparable results are

achieved over several data sets from the UC Irvine Machine Learning Database Repository, with an average increase in accuracy from 90.7% and 91.3% using optimized SSE and CE networks, respectively, to 92.1% for CB1. Analysis indicates that CB1 performs a fundamentally different search of the feature space than optimizing SSE or CE and produces significantly different solutions.

1 Introduction

Artificial neural networks have received substantial attention as robust learning models for applications involving classification and function approximation (Rumelhart, Hinton, & Williams, 1985). This work proposes the use of classification-based (CB) objective functions to improve backpropagation, increasing generalization on complex classification tasks. The CB1 algorithm is presented as the main contribution. It is an example of a CB objective function suited to learning classification tasks. CB1 seeks to directly minimize classification error by backpropagating error only on misclassified patterns from output nodes that are responsible for the misclassification. In doing so, it updates the network parameters as little as possible. This technique discourages weight saturation and overfitting and is conducive to higher accuracy in classification problems than optimizing with respect to common error functions, such as sum-squared error (SSE) and cross-entropy (CE).

CB1 is shown to perform markedly better on a large OCR data set than an optimized backpropagation network learning with respect to SSE or CE, increasing classification

accuracy from 97.86% and 98.10%, respectively, to 99.11%. Comparable increases in accuracy are achieved on several classification problems from the UC Irvine Machine Learning Repository, with an average increase in accuracy from 90.7% and 91.3% for optimized SSE and CE networks, respectively, to 92.1% for CB1 performing 10-fold stratified cross-validation. Analysis indicates that CB1 performs a fundamentally different search of the feature space than backpropagation optimizing SSE or CE and produces significantly different solutions.

A background discussion and comparison of common objective functions to CB1 is provided in Section 2. The CB1 heuristic is presented in Section 3. Experiments and analysis are given in Section 4 and discussion in Section 5. Further discussion of learning issues with feed-forward backpropagation neural networks, overfitting, and how these relate to CB1 is presented in Section 6. Future work is outlined in Section 7 and conclusions are presented in Section 8.

2 Objective Functions

Since gradient descent procedures, such as backpropagation, do not allow direct minimization of the number of misclassified patterns (Duda, Hart, & Stork, 2001), an error or objective function must be derived that results in increased classification accuracy as objective error is minimized. Network output values must have a corresponding error measure derived by their deviance from target output values. Quantifying the output error provides a way for iteratively updating the network weights

in order to minimize that error and thereby achieve more accurate output. However, error functions do not always decrease monotonically with the classification error, which is the real goal of the learner.

Classification of N classes is often viewed as a regression problem with an N-valued response, with a target value of 1 in the n^{th} position if the observation falls in class n and 0 otherwise (LeBlanc & Tibshirani, 1993). The values of zero and one can be considered idealized or hard target values. However, in practice there is no reason why class targets must take on these values.

To generalize well, a network must be trained using a proper objective function. Backpropagation training often uses an objective function that encourages making weights larger in an attempt to output a value approaching hard targets of 0 or 1 (±1 for the *htan* function). Using hard targets is a naive way of training and creates several practical problems. Different portions of the data are learned at different times during training, and using hard targets not only often leads to premature weight saturation, making it harder and slower to learn patterns that have yet to be learned, but also forces the learner to overfit on patterns that have already been learned. One conventional approach uses "softer" targets like 0.1 and 0.9. This presents a less severe solution but still suffers from overfitting.

Rankprop (Caruana, Baluja, & Mitchell, 1996) provides an alternative method to training with hard target values and empirically shows that it improves generalization. Rankprop

records the output of the learner for each training pattern. It then sorts the patterns in the training set based on class, then according to output values. Thus, a rank of the patterns consistent with the current model is developed and used to define the target values on the next epoch. The idea behind Rankprop is that in the case of complex nonlinear solutions a simpler, less nonlinear function is provided to learn instead. The resulting simpler model often generalizes better. CB1 also provides a simpler function for the network to learn that leads to better generalization.

2.1 Conventional objective functions

The validity of using common differentiable measures like SSE as an objective function to minimize error relies on the assumption that pattern outputs are offset by inherent gaussian noise, being normally distributed about a cluster mean. For approximating the function of an arbitrary signal this presumption often holds. However, this assumption is invalid for classification tasks, where assigned real-valued target vectors are arbitrary values used to represent class labels. This suggests that other error metrics are more suited to classification problems.

In (LeCun, Denker, & Solla, 1990), a study of the *digits* problem revealed that heuristically reducing the number of network parameters by a factor of two increased training set MSE by a factor of ten, while generalization MSE increased by only 50%, and test set classification error actually decreased. This suggests that minimizing MSE might not be a reliable objective function for complex classification tasks. This also implies that comparison studies showing "improvements" through a reduction of

SSE/MSE on classification tasks are not significant unless classification accuracy increases likewise.

Cross-entropy (CE) assumes idealized class outputs (i.e., target values of zero or one for a sigmoid activation) rather than noisy outputs as does SSE (Mitchell, 1997) and is therefore more appropriate to classification problems. The classification figure-of-merit (CFM) objective function was introduced in (Hampshire II, 1990) for learning classification problems when it was shown that SSE and CE errors are not necessarily correlated with classification accuracy. CFM separates the values of network outputs by as large a range as possible such that error minimization is monotonic with increasing classification accuracy. Like SSE and CE, this metric encourages weight saturation, which is often indicative of overfitting and detrimental to generalization (Bartlett, 1998).

2.2 Classification-based objective functions

Generalizing well, not minimizing error with respect to an objective function, is the goal of learning. LeCun's *digits* study mentioned above illustrates how objective functions can inaccurately reflect how well a problem has been learned. Hence, the objective function chosen for learning should approximate the true goal of the learner as closely as possible. CB1 more directly portrays how well the network has learned to classify the training patterns.

Similar to CFM, CB1 also attempts to increase the range between output activations. However, CB1 widens the range between outputs only when there is classification error.

When a misclassification is made, error is backpropagated only from those outputs that are credited with producing the error. Observe that this effectively narrows the gap between outputs, as they are transposed with respect to their traditional 0-1 target values. This approach allows the network to relax more conservatively into a solution and discourages weight saturation and overfitting.

3 CB1: A Classification-based Error Heuristic

There is an inherent tradeoff between fitting the (limited) training data sample perfectly and generalizing accurately on the entire population (see Section 6.4). There are several possible ways to process the network's output vector in calculating an error signal for backpropagation to fit the data properly. A simple variant involves modifying the objective function by providing a maximum error tolerance threshold, d_{max}, which is the smallest absolute output error to be backpropagated. In other words, given $d_{max} > 0$, a target value, t_j, and network output, o_j, no network update occurs if the absolute error $| t_j - o_j | < d_{max}$. This threshold is arbitrarily chosen to represent a point at which a pattern has been sufficiently approximated. With an error threshold, the network is permitted to converge with smaller weights (Schiffmann, Joost, & Werner, 1993). More dynamic approaches, such as Rankprop (Caruana, 1995), avoid the use of pre-defined "hard" targets, setting ranked "soft" target values for the training patterns each epoch.

CB1, introduced here, considers the entire output vector of the network to determine the error of each output node. For each pattern considered, CB1 backpropagates error

through the network only on misclassified patterns. As this technique forces networks to learn only when explicit evidence is presented that their state is a detriment to classification accuracy, we have called the approach classification-based training. Like Rankprop, CB1 avoids the use of hard target values. However, rather than providing soft targets, it avoids the use of predetermined target values altogether. The objective of CB training is *not* to minimize the error between target and output values, but rather to produce output values that can be accurately translated to correct classifications. With CB training, smaller weights near zero can provide an acceptable solution for classification tasks. Keeping weights smaller avoids problems caused by weight saturation.

Network weights are updated during CB training exclusively to minimize classification error. When the network misclassifies a pattern, credit for the error is assigned to two sources. The first is the set of output nodes with higher output values than the target class node (resulting in the system outputting the wrong class value). The second is the target output node itself, which outputs a value too low to produce the correct classification. This approach is formalized as follows.

3.1 CB1 error function

Let N be the number of output nodes in a network. Let o designate the activation value of a node ($0 \le o \le 1$ for sigmoid). Let o_k be the activation value of the k^{th} output node in the network ($1 \le k \le N$). Let T designate the target class for the current pattern and c_k signify the class label of the k^{th} output node. For target output nodes, $c_k = T$, and for non-target

output nodes, $c_k \neq T$. Non-target output nodes are called *competitors*. Often, class labels are indicated in training by setting the target value of one output node high and setting the rest low. This restriction is not made here, as it is possible for more than one output node to act as a target node for a class label in the general case. However, for the remaining discussion standard 1-of-*N* target designations are assumed.

Let $o_{T\max}$ denote the value of the highest-outputting target output node, or formally

$$o_{T\max} \equiv \max \{ o_k : c_k = T \}.$$

Let $o_{\sim T\max}$ denote the value of the competitor outputting the highest o,

$$o_{\sim T\max} \equiv \max \{ o_k : c_k \neq T \}.$$

The error, ε, back-propagated from the k^{th} output node is then defined as

$$\varepsilon_k \equiv \begin{cases} o_{\sim T\max} - o_k & \text{if } c_k = T \text{ and } (o_{\sim T\max} \geq o_{T\max}) \\ o_{T\max} - o_k & \text{if } c_k \neq T \text{ and } (o_k \geq o_{T\max}) \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

The error (1) represented in closed form is

$$\varepsilon_k \equiv (o_{\sim T\max} - o_k)\mathrm{I}(c_k = T \text{ and } (o_{\sim T\max} \geq o_{T\max})) + (o_{T\max} - o_k)\mathrm{I}(c_k \neq T \text{ and } (o_k \geq o_{T\max}))$$

where I is the indicator or characteristic function. Thus, a target output generates an error signal only if there is some competitor with an equal or higher value than $o_{T\max}$, signaling a potential misclassification. Non-target outputs likewise generate an error signal only if they have an output equal or higher than $o_{T\max}$, indicating they are responsible for the misclassification. The intuitive rationale behind this is that if the error is continually reduced on misclassified patterns, they will eventually be classified correctly.

The error delta used for backpropagation is

$$\delta_k = \varepsilon_k f'(o_k)$$

where $f'(o_k)$ is the standard error gradient, which is

$$f'(o_k) = o_k (1 - o_k)$$

for a sigmoid squashing function, and can be removed on output nodes when using cross-entropy (Joost & Schiffmann, 1998).

To illustrate how CB training works, consider a three-class problem. For a particular pattern, assume that the third class is the target. Traditionally, this translates into a target vector of (0, 0, 1). Assume that on this pattern, a 3-output network outputs (0.1, 0.2, 0.4). While the third output (the target) has substantial squared error (0.36), the first two output

values (the competitors) are sufficiently low, enough so that it is possible to extract the correct classification (the third class is chosen since its value is highest). Since the pattern is classified correctly, the network parameters remain unchanged.

Only if one of the competitors outputs a higher value than the target would a non-zero error signal be backpropagated from any of the output nodes. In the case that the network outputs (0.1, 0.4, 0.3), both the second and third output nodes would backpropagate error: the second since it outputs higher than the target node, and the third, since a competitor outputs a higher value than it. The error signal is set at the minimum amount possible to produce a correct classification.

3.2 Advantages of CB training

When a pattern is already classified correctly, forcing output values closer to 0 or 1 often results in weight saturation and overfitting. This needlessly increases network variance (sensitivity to the training data), increasing classification error on test data. Training without idealized or predetermined target outputs allow a pattern to be potentially "learned" with any target node output, providing competitors output lower values. This insight is the driving motivation behind CB training, which avoids this practice.

CB training of a network proceeds at a different pace than optimizing SSE or CE as the objective function. Weights are updated only through necessity. Backpropagating a non-zero error signal from *only* the outputs that directly contribute to classification error results in considerably fewer weight updates overall (observe that this number is

proportional to the classification accuracy) and allows the model to relax more gradually into a solution. CB training learns only as much as required to remove misclassifications and thereby discourages overfitting. This approach is reminiscent of training with an error threshold; however, whereas a fixed error threshold causes training to stop at a pre-specified point, meaning weights must increase to a magnitude sufficient to achieve this threshold, CB training dynamically halts learning at the *first possible point* that correctly classifies a training pattern. This can be considered an implementation of a *dynamic error threshold* that is unique to each training pattern and network state.

3.3 Increasing the margin with CB training

Figure 1 illustrates how sample variance in the training set can influence the decision surface arrived at using SSE and CB1 error functions on a two-class problem. Overfitting is minimized in CB training because outliers (noisy patterns) have minimal detrimental impact to the decision surface's accuracy. This is because the target output is only required to output a value negligibly higher than the highest competitor before the training process stops updating the network parameters. In Figure 1b, the CB1 decision surface remains next to a noisy pattern. This is in contrast to conventional SSE training, where hard target values of 0 and 1 require pushing the decision surface as far away from all training points as possible, including noisy outliers (see Figure 1a). Hence, test patterns in the area near the question mark falling next to an outlier of the competing class have a better chance of being correctly classified and network variance is substantially reduced.

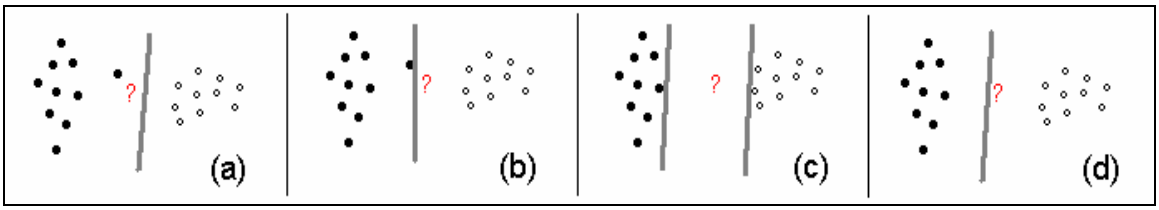

Figure 1. Typical decision boundaries for SSE networks (a,d), CB1 networks without an error margin (c), and CB1 with a small error margin (b,d). CB1 induces a boundary more robust to noisy patterns.

When CB training, it is common for the highest outputting node in the network, which we will call o_{max}, to output a value only slightly higher than the second-highest-outputting node (see Figure 2). This is true for correctly classified patterns (those above 0 in Figure 2), and also for misclassified ones (those below 0). This means that most training patterns remain physically close to the decision surface throughout training. In the absence of outliers, then, one would expect the heuristic to arrive at a decision surface similar to those portrayed in Figure 1c. According to the application this might not be desirable.

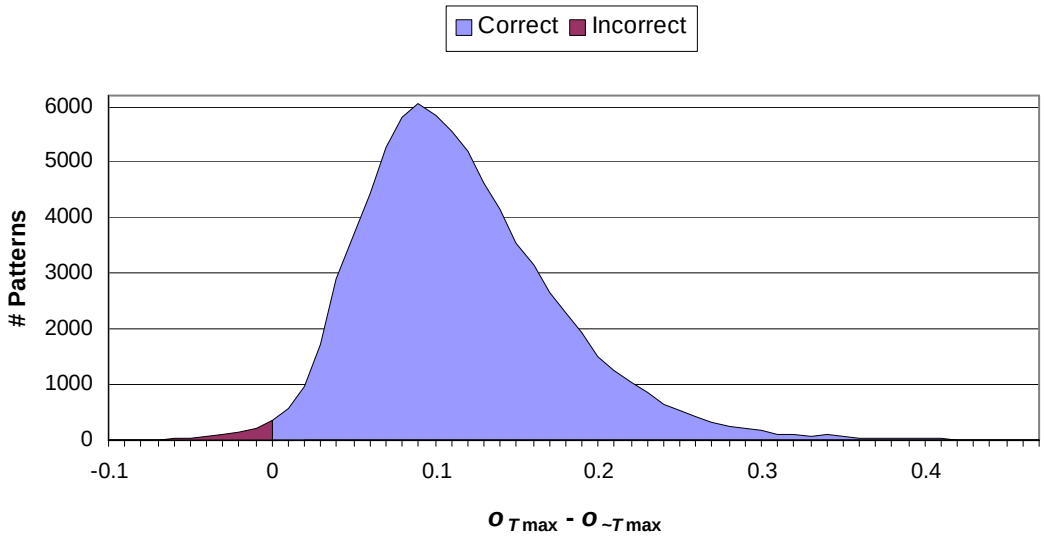

Figure 2. Network output error margin after CB training on *OCR* data set. Values of output nodes are typically very close together, yet nearly all patterns are correctly classified.

An error margin, μ, can be introduced during training that serves as a confidence buffer between the outputs of target and competitor nodes. The value for μ can range from –1 to +1 under the sigmoid function. For no error signal to be backpropagated from the target output, an error margin requires that $o_{\sim T\max} + \mu < o_{T\max}$. Conversely, for a competing node k with output o_k, the inequality $o_k < o_{T\max} - \mu$ must be satisfied for no error signal to be backpropagated from k. This augmentation to (1) is presented as

$$\varepsilon_k \equiv \begin{cases} \min(o_{\sim T\max} + \mu - o_k, 1) & \text{if } c_k = T \text{ and } (o_{\sim T\max} + \mu \geq o_{T\max}) \\ \max(o_{T\max} - \mu - o_k, -1) & \text{if } c_k \neq T \text{ and } (o_k \geq o_{T\max} - \mu) \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

where min(·,1) and max(·,0) enforce the [-1,1] error range of the logistic function. In this way, CB1 with a small μ (e.g. 0.1) approximates the SSE solution and the margin is maximized even in the absence of outliers (see Figure 1d).

During the training process, the value of μ can be altered and might even be negative to begin with, not expressly requiring correct classification at first. This gives the network time to configure its parameters in an even more uninhibited fashion. Then μ is increased to an interval sufficient to account for the variance that appears in the domain data, allowing for robust generalization. The value of μ can also be decreased, and remain negative as training is concluded to account for noisy outliers. A preliminary analysis of updating μ during training has shown promise (Rimer & Martinez, 2004).

Including a margin also decreases the amount of "classification oscillation" that occurs as outputs react to one another. When $\mu = 0$, patterns remain close to the decision surface during training. As training proceeds and the decision surface shifts around, patterns frequently slide back to the wrong side of the decision surface. Introducing a small, positive μ requires patterns to be situated further away from the decision surface and reduces the incidence of renewed misclassification, leading to quicker convergence. Observe that at the extreme value of $\mu = 1$, CB1 reverts to standard SSE training, with target values of 1.0 and 0.0 required for all positive and negative classes, respectively.

4 Experiments and Analysis

Neural networks were trained through backpropagation optimizing SSE and CE, and through CB1 to explore empirical advantages of CB training techniques. These models include:

- A single-output network on two-class problems (positive patterns are assigned a target value of 1.0, negative patterns are assigned a target value of 0.0)
- A single *N*-output network (one output per class on multi-class problems)
- *N* independent single-output networks on multi-class problems (one per class)

Experiments were conducted over a variety of data sets with varying characteristics, differing by:

- Size of data set (150 instances to half a million)
- Number of features (two to hundreds)
- Number of labeled data classes (two to forty-seven)
- Complexity of data distribution (nearly linearly separable to highly complex)

Problems were drawn from the UC Irvine Machine Learning Database Repository (UCI MLR) (Blake & Merz, 1998) and from a large database of machine printed characters gathered for OCR. This provides a vantage point from which to evaluate the robustness of CB1.

In empirical comparisons among different learning methods, appropriate training parameters were determined to optimize each model. For further conceptual analysis and illustration of the behavior of these systems, results of experiments using a range of parameters are provided.

4.1 Data sets

The performance of SSE versus CB training has been evaluated on an OCR data corpus (*OCR*) consisting of roughly 370,000 alphanumeric character patterns and 47 symbolic classes, partitioned into 277,000 training patterns, 31,000 holdout set patterns, and 62,000 test patterns. Results on this data set were first presented in (Rimer, Andersen, & Martinez, 2001a).

Two network topologies were evaluated for learning *OCR*, a single *N*-output network and *N* single-output networks (*N* = 47 for *OCR*).

Additionally, eight well-known classification problems were selected from the UCI MLR. Descriptions of the selected data sets are listed as follows:

> **ann** – 7200 instances with 15 binary and 6 continuous attributes in 3 classes. The task is to determine whether a patient referred to the clinic is hypothyroid.

bcw – 699 instances with 9 linear attributes in 2 classes. The task is to detect the presence of malignant versus benign breast cancer.

ionosphere – 351 instances with 34 numeric attributes in 2 classes. This data set classifies the presence of free electrons in the ionosphere.

iris – 150 instances with 4 numeric attributes in 3 classes. This classic machine learning data set classifies the species of various iris plants based on physical measurements.

musk2 – 6598 instances with 166 continuous attributes in 2 classes. The task is to predict whether new molecules will be musks or non-musks.

pima – 768 instances with 8 numeric attributes in 2 classes. The predictive class in this data set is whether or not the tested individual has diabetes.

sonar – 208 instances with 60 continuous attributes in 2 classes. The task is to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock.

wine – 178 instances with 13 continuous attributes in 3 classes. The attributes give various parts of the chemical composition of the wine and the task is to determine the wine's origin.

A single network with one output per class was used to learn these problems. Results on UCI MLR problems were averaged using 10-fold stratified cross-validation.

4.2 Training parameters

Experiments were performed comparing the SSE, CE, and CB1 objective functions. Fully connected feed-forward networks with a single hidden layer trained through standard on-line backpropagation were used. In all experiments, weights were initialized to uniform random values in the range [-0.1,0.1]. Networks trained to optimize SSE and CE used an error tolerance threshold (d_{max}, described in Section 2) of 0.1.

Feature values (both nominal and continuous) were normalized between zero and one. Training patterns were randomly shuffled before each epoch. For each simulation, a random seed for network weight initialization and pattern shuffling was used across all networks tested.

Network learning parameters on *OCR* for each error function have been extensively optimized over the course of hundreds of empirical evaluations, comprised of tests of networks with topologies of one to three hidden layers, ten to five hundred hidden nodes per layer, learning rates from 0.01 to 0.5, and momentum from 0.0 to 0.99. We performed two sets of experiments on *OCR*. One tested multi-task learning (MTL), or using a single network with multiple output nodes, and the other used a separate network to learn each problem class. Pattern classification was determined by *winner-take-all* (the class of the highest outputting node is chosen) on all models tested.

For experiments presented here evaluating MTL, a hidden layer of 128 nodes was used. For experiments presented here training a separate, single output node network for each

class label, each network has a single hidden layer of 32 hidden nodes. The learning rate was 0.1 and momentum was 0.7. Training was halted after 50 epochs (nearly 14 million patterns) of seeing no improvement in accuracy on the holdout set. The model selected for testing was the one with the best holdout set classification accuracy. Although we only present empirical results for this combination of parameter values, it is noted that relative accuracies among the error functions were typical and comparable over the range of learning parameters, network sizes and topologies tested.

On UCI MLR data sets, network size was optimized to maximize generalization for each problem and error function. Optimized numbers of hidden nodes used for learning UCI MLR data sets are listed in Table 1. Learning rate was 0.1 and momentum was 0.5 for all UCI MLR problems. Training continued until the training set was successfully learned or until holdout error ceased to decrease for 500 consecutive epochs. The model selected for testing was the one with the best holdout set classification accuracy.

Table 1. Network architectures on MLR problems.

The number of input, hidden, and output nodes per network is shown.

Data set	SSE Network	CE Network	CB1 Network
ann	21-30-3	21-30-3	21-30-3
bcw	9-15-2	9-25-2	9-10-2
ionosphere	34-7-2	34-9-2	34-9-2
iris	4-1-3	4-1-3	4-1-3
musk2	166-5-2	166-5-2	166-5-2
pima	8-8-2	8-8-2	8-16-2
sonar	60-15-2	60-5-2	60-15-2
wine	13-16-3	13-8-3	13-16-3

4.3 Results

Empirical results on *OCR* and the UCI MLR data sets are presented in the following sub-sections.

4.3.1 OCR data set

Tables 2 and 3 display the results of standard SSE and CE backpropagation versus CB1 on *OCR*. *Train %* and *Test %* are the training and test set accuracy on the selected network model in percent, averaged over five training runs. How well each model generalizes is indicated by *Test % / Train %*. *Train MSE* and *Test MSE* are the mean squared errors for the training and test sets on the epoch from which this model was chosen. Best values are printed in bold face.

Table 2. Results on *OCR* with *N* single-output node networks.

Error function	Train %	Test %	Test %/Train %	Train MSE	Test MSE
SSE	99.28	97.86	.9857	**.0047**	**.0092**
CE	99.37	98.10	.9872	.0094	.0110
CB1 ($\mu = 0.05$)	**99.61**	**99.11**	**.9950**	.1830	.2410

Table 3. Results on *OCR* with one *N*-output node network.

Error function	Train %	Test %	Test %/Train %	Train MSE	Test MSE
SSE	98.79	98.62	.9983	.0274	.0313
CE	99.09	98.91	.9982	**.0160**	**.0221**
CB1 ($\mu = 0$)	99.15	98.96	.9981	.1594	.1800
CB1 ($\mu = 0.1$)	**99.38**	**99.26**	**.9988**	.0992	.0968

These tests demonstrate that multi-task learning of *OCR* generalizes better than using a separate network to learn each problem class with SSE and CE objective functions. Even though training accuracy is lower on the SSE and CE multi-output networks than

multiple networks, generalization (i.e., test accuracy / train accuracy) is improved. Observe that with a single multiple-output network, test set accuracy is within a fraction of one percent of training set accuracy using any of the tested error functions. This occurs since little overfitting can occur in this size network when attempting to learn all classes simultaneously. Worse generalization was observed using networks with more hidden nodes. When training a separate network for each class, each network has much greater potential to overfit since there are many more network parameters. This behavior is exhibited to lesser degree with CB training.

Optimizing CE on *OCR* trains and generalizes better than SSE, and CB1 performs significantly better than both of these ($p < 0.0001$). Network models generated with CB1 also have improved generalization. Observe that CB1 has a much higher MSE than the other methods, yet overfitting is reduced and generalization is improved. This is an important point that is discussed further in Section 5.

Generalization with the best CB1 model is 0.73% greater than the best model trained with SSE and 0.53% greater that the best CE-trained model. Considering only multiple-output networks, error drops from 1.38% for SSE and 1.09% for CE to 0.74% for CB1, increases in accuracy of 0.64% and 0.35%, respectively ($p < 0.0001$). Considering only the multiple single-output network models, error drops from 2.14% with SSE and 1.90% with CE to 0.89% with CB1, increases in accuracy of 1.25% and 1.01%, respectively ($p < 0.0001$).

4.3.2 UCI MLR data sets

Table 4 lists the results of a naïve Bayes classifier taken from (Zarndt, 1995), standard SSE and CE backpropagation, and CB1 with SSE and CE error updates (whether the error gradient, discussed in Section 3.1, is $o(1-o)$ or 1, respectively) on eight UCI MLR classification problems. Results were gathered using 10-fold stratified cross validation and averaged over thirty randomly-initialized training runs.

The first value in each cell is the average classification accuracy of the selected model. The second value is the standard deviation over all runs. The best generalization for each problem is bolded and the second best value is italicized. An asterisk indicates a confidence within $p < 0.05$ that the highest accuracy is significantly better than the second highest. The last two columns indicate the difference in value between CB1 (with SSE or CE error gradient) and SSE and CE optimization. Here, a higher first value in each cell indicates greater improvement, and a lower second value indicates smaller standard deviation.

Table 4. Results on selected data sets from UCI MLR using 10-fold stratified cross-validation. Best values are shown in bold and second best in italics. Statistical significance of $p < 0.05$ of the most accurate algorithm is signified by an asterisk.

Data set	Bayes	SSE	CE	CB1 SSE	CB1 CE	CB1 SSE – SSE	CB1 CE – CE
ann	**99.7***	98.25	98.33	97.62	*98.76*	-0.63	0.43
	0.1	0.54	0.53	0.47	*0.51*	-0.07	-0.02
bcw	93.6	96.96	97.06	*97.22*	**97.36***	0.26	0.30
	3.8	2.01	**1.81**	2.01	**1.81**	0.0	0.0
ionosphere	85.5	89.00	*90.80*	90.60	**90.88**	1.60	0.08
	4.9	4.72	4.64	**3.75**	*3.87*	-0.97	-0.77
iris	94.7	93.83	94.37	**95.47***	*95.37**	1.64	2.00
	6.9	5.68	5.87	*5.31*	**5.25**	-0.37	-0.62
musk2	97.1	99.06	98.56	*99.15*	**99.27**	0.09	0.71
	0.7	0.37	0.62	*0.36*	**0.29**	-0.01	-0.33
pima	72.2	76.26	76.11	*76.69**	**76.82***	0.43	0.71
	6.9	*4.24*	4.36	**3.43**	6.46	-0.81	2.10
sonar	73.1	76.06	78.87	*80.77*	**81.92***	4.71	3.05
	11.3	9.37	9.03	*9.02*	**8.60**	-0.35	-0.43
wine	94.4	96.29	96.74	**98.31***	*97.19*	2.02	0.45
	5.9	4.45	4.13	**3.49**	*3.47*	-0.96	-0.66
Average	88.79	90.69	91.35	*91.97*	**92.20**	**1.28**	**0.85**
	5.06	3.92	3.87	**3.48**	*3.79*	**-0.44**	**-0.08**

The average increase in classification accuracy is from 90.69% for SSE training to 91.97% for CB1 with SSE error gradient, a 1.28% decrease in error (significant to $p < 0.035$). Using CB1 with a CE error gradient demonstrated a 0.85% increase in accuracy over CE training, from 91.35% to 92.20%. An overall decrease in standard deviation also indicates that CB training is more robust to initial parameter values and pattern variance then SSE and CE optimization. This supports the hypothesis that weight saturation and overfit is reduced and generalization is improved by CB training.

5 Discussion

Standard backpropagation and other gradient descent learning techniques do not consider or attempt to maximize the number of correctly classified training patterns (Duda, Hart, & Stork, 2001). CB1 incorporates a more direct minimization of misclassified patterns in gradient descent procedures by reducing error on only misclassified patterns.

Since CB1 does not train using ideal target values like SSE and CE, MSE drops very slightly as training accuracy is improved (see Figure 3a). This is in contrast to the strong, immediate drop in MSE illustrative of standard SSE optimization (see Figure 3b). CE displays similar behavior to SSE optimization but is omitted from the following discussion for brevity.

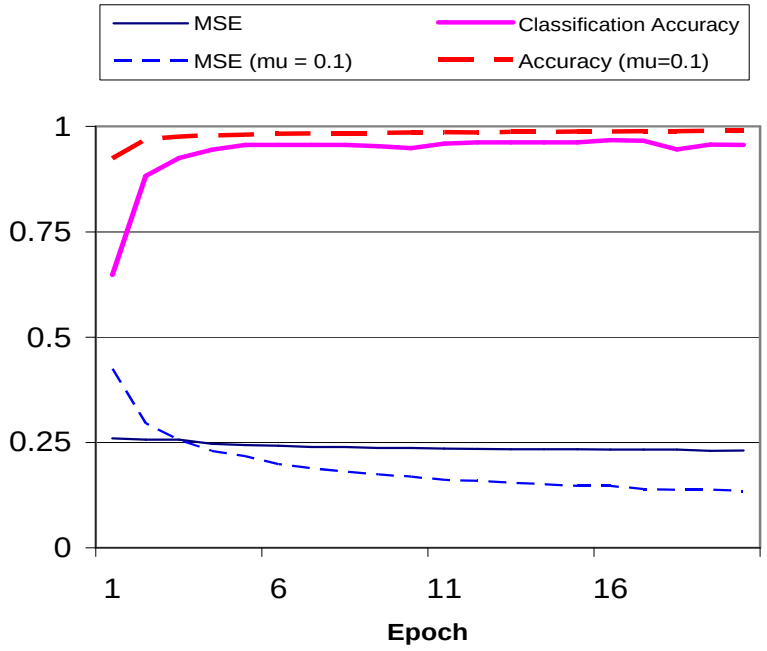

Figure 3a. Classification accuracy and MSE during CB1 training.

MSE decreases very slowly during training.

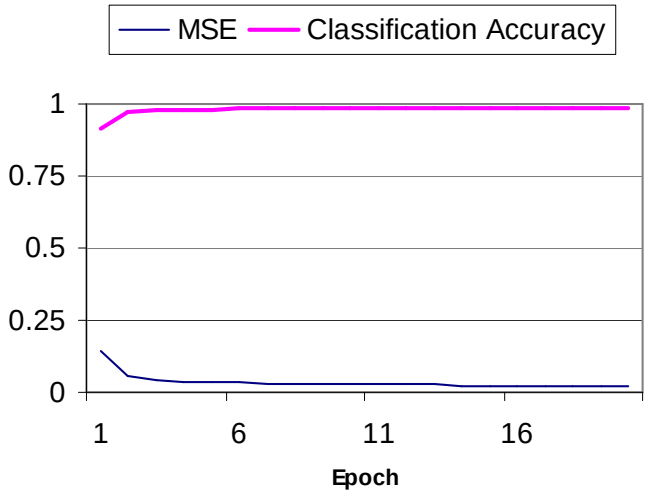

Figure 3b. Classification accuracy and MSE during SSE optimization.

MSE decreases quickly during training.

In Figure 3a, rather than converging to zero, MSE remains just under 0.25 as training progresses with μ=0. With μ=0.1, MSE decreases to around 0.15. A large MSE is incurred by pattern outputs being very far away from the conventional 0-1 target values. Observe that a MSE of 0.25 is equivalent to a mean error of 0.5, which illustrates that output activations are close to 0.5 on average throughout training. This indicates that the weights for these outputs are close to zero. This suggests that CB training performs a fundamentally different search in feature space than standard SSE/CE optimization. It descends towards different minima and converges to a feature location physically distant from SSE/CE solutions. This also indicates that high-accuracy solutions exist where SSE are CE are about as high as when training starts on a network initialized to small random weights.

Figures 4-6 give insight into the behavior of the network during the learning process using four error functions. The surface plot shows a histogram of the values output by the network output nodes on the training patterns every epoch (Figure 4) and every tenth training epoch (Figures 5 and 6). Figures 4, 5a and 5b show learning minimizing SSE, and Figures 6a and 6b show behavior during CB1 training. The results shown here are for the *bcw* data set, but such behavior is generally representative of all data sets tested.

In Figure 4, it can be seen that SSE training forces the network to output values approaching 0 and 1 in fewer than ten epochs. It is noted that even after the first epoch, network outputs are already completely separate and distinct. Using a d_{max} of 0.1 reduces

this tendency somewhat. Observe the flattened peaks for positive patterns in Figure 5b that do not exist in 5a.

CB training produces a starkly different behavior. In Figure 6a, it can be observed that all patterns output around 0.5 during the entire training process. In Figure 6b, incorporating a confidence margin of $\mu = 0.1$ widens the spread of output values, causing the output clusters of the two classes to visibly split apart as training progresses.

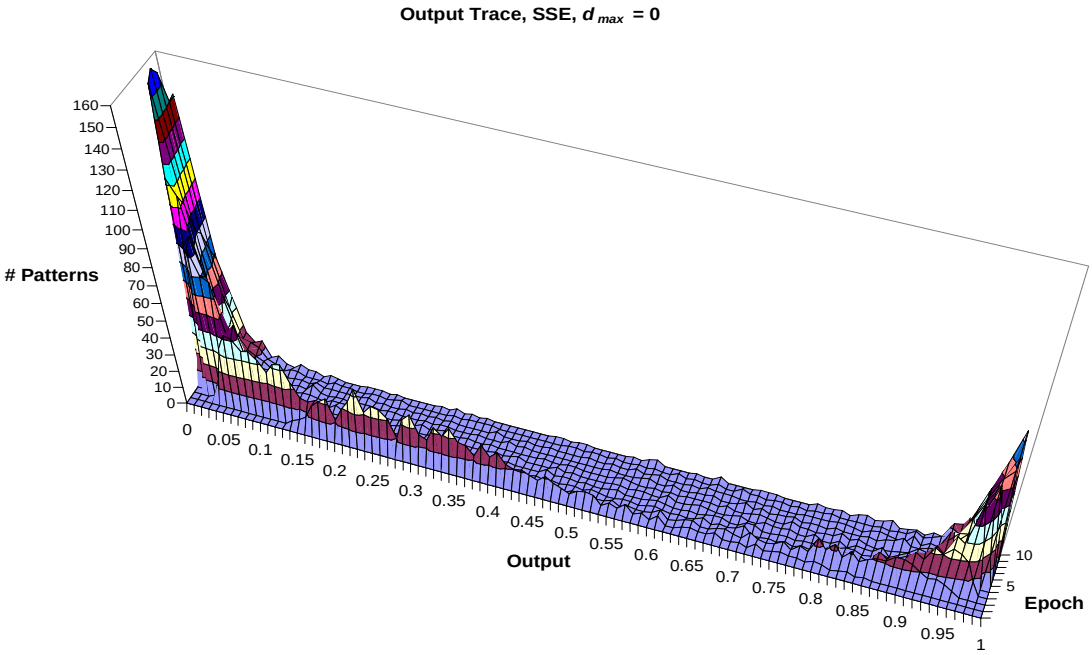

Figure 4. Network output trace SSE optimization on *bcw* (first ten epochs). Weight saturation occurs after only a few training epochs.

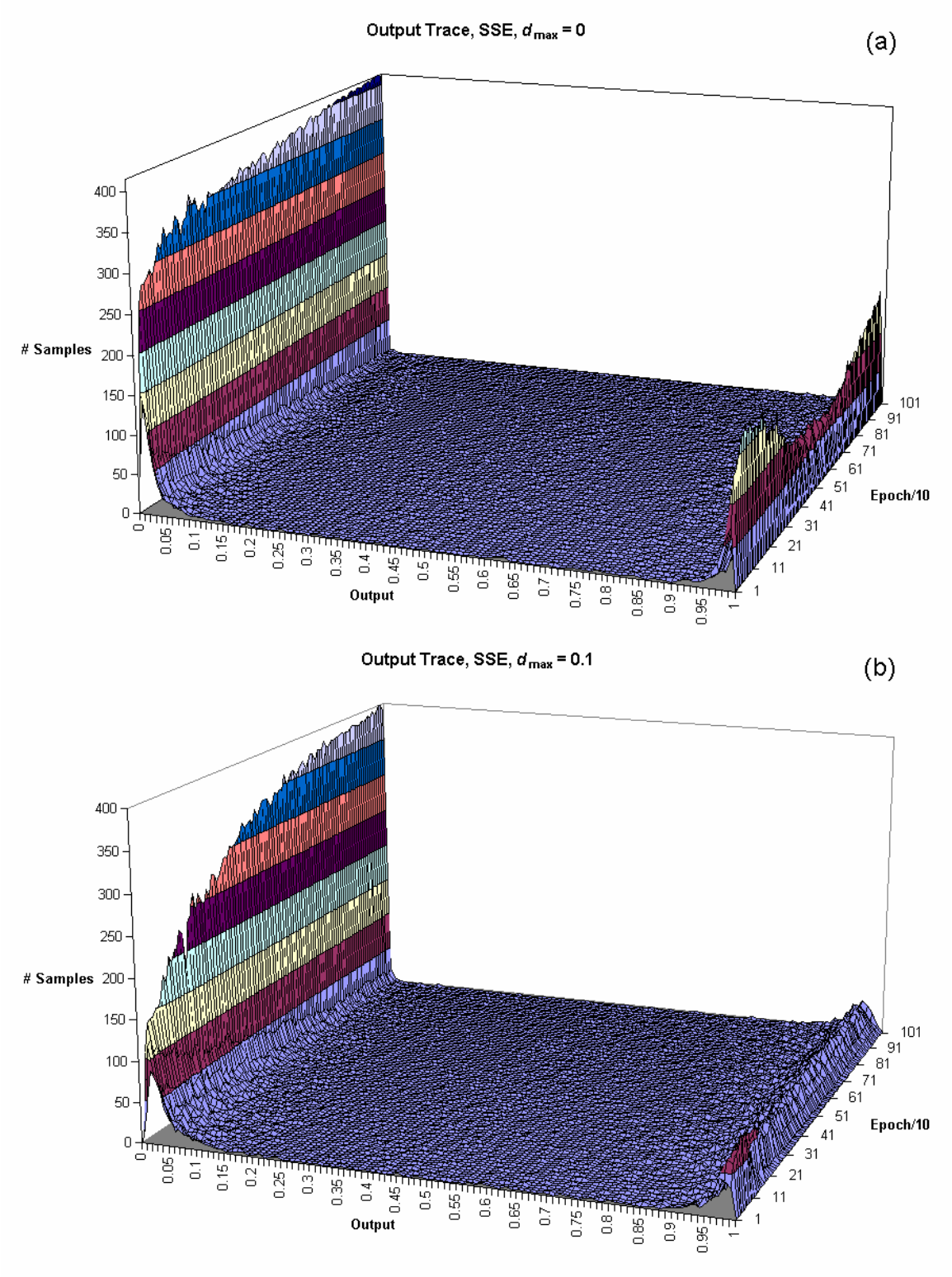

Figure 5. Network output trace during SSE optimization on *bcw*.

Network weights become quickly saturated during training.

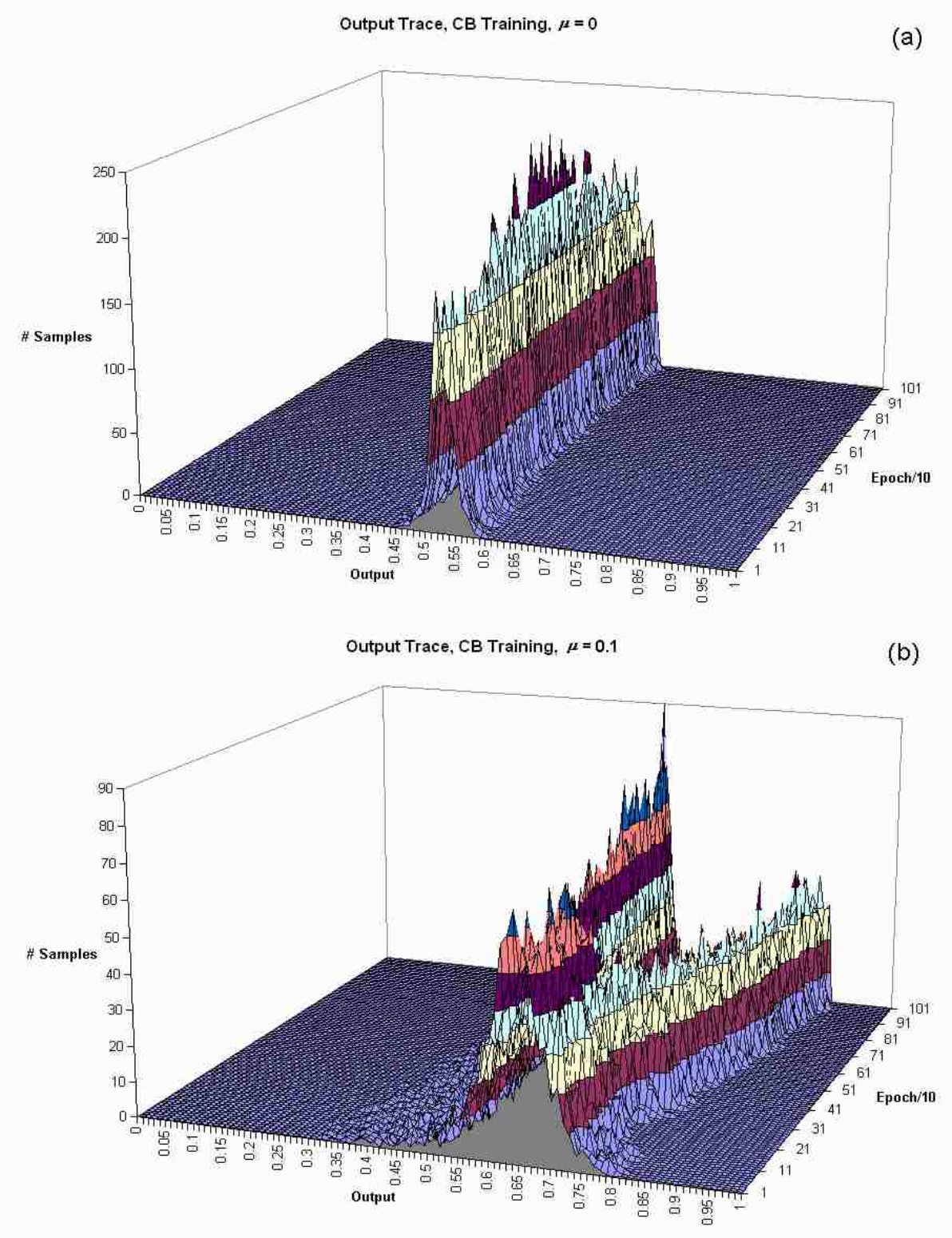

Figure 6. Network output trace during CB training on *bcw*.

Network weights do not become saturated during training.

5.1 Empirical effects of an error margin

Figure 7 depicts the results of training with CB1 on *bcw* with values for μ ranging from 0 to 0.9. Each value bar shown is the averaged classification accuracy with standard deviation using 10-fold stratified cross validation.

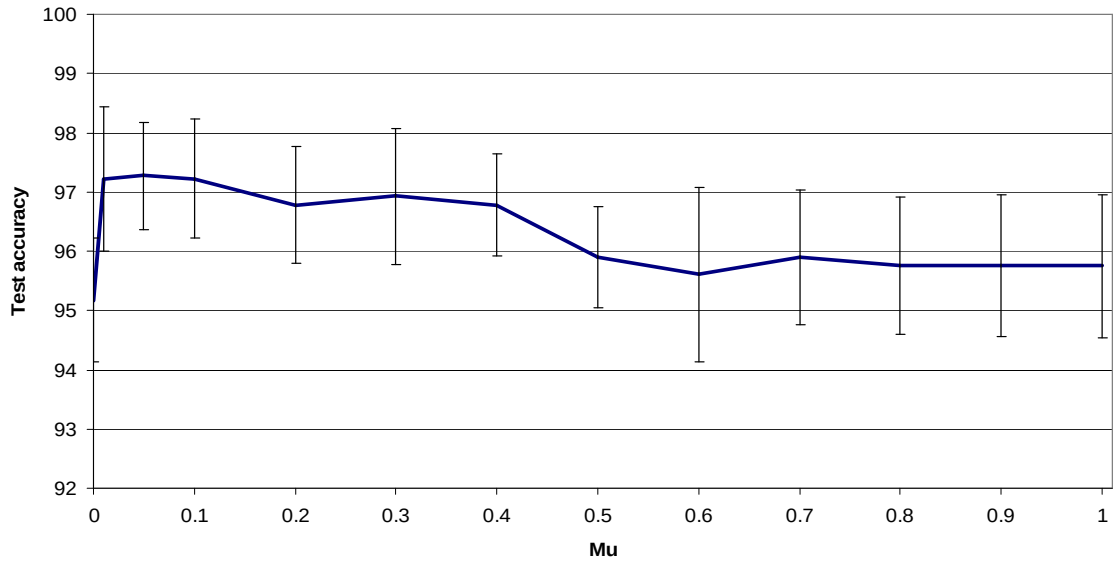

Figure 7. 10-fold CV results for CB training on *bcw* with μ.

Small, non-zero values for μ typically demonstrate the best generalization.

This shows that CB1 is fairly robust to the selection of μ. Values for $\mu > 0$ cause the decision surface to be more removed from proximal test patterns than when $\mu = 0$ and have better generalization. Values for μ closer to 0 show the most improvement and values closer to 1 cause CB1 to revert proportionally to the behavior of standard SSE minimization. (Note that the accuracies shown for $\mu \sim 1.0$ do not match the accuracy for

SSE training in Table 4 because the accuracies in Table 4 are based on roughly optimized parameters for each error function, and CB and standard training have different optimal learning parameters.)

5.2 Effect of SSE on output values

Following a training run on *OCR* training to minimize SSE, winning network outputs on the test set were distributed as shown on the logarithmic scale in Figure 8. The network outputs were very close to 1.0 on the majority of the patterns. Only 2-3% of the patterns lie close to where the decision surface is located (implicitly at 0.5). The weights have grown in magnitude to the point that the dividing sigmoidal surface is very sharp.

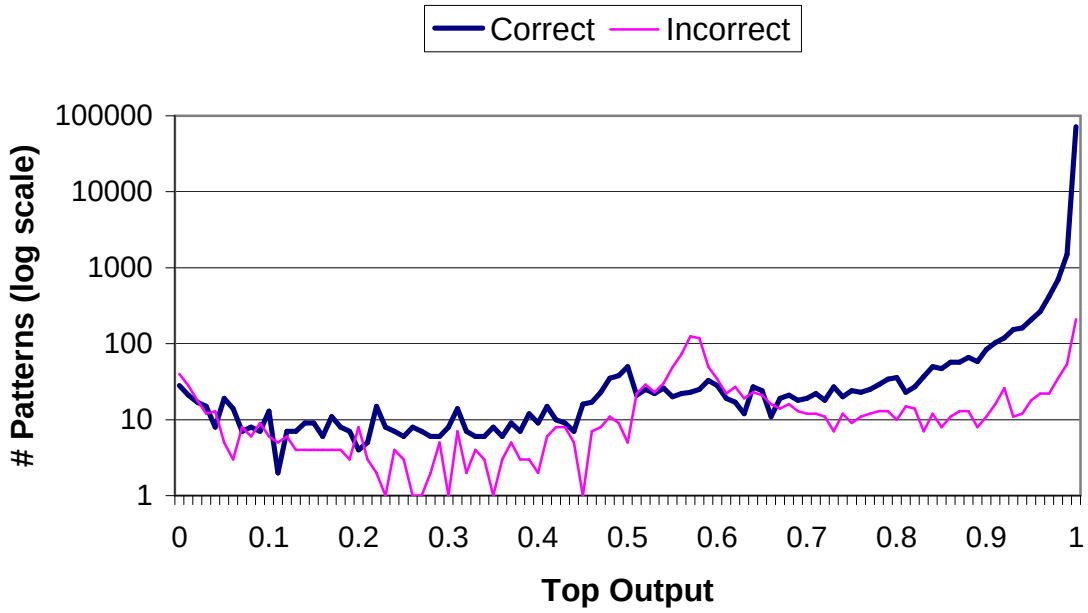

Figure 8. Network outputs on *OCR* test set after SSE minimization are typically close to 1, indicative of large weights.

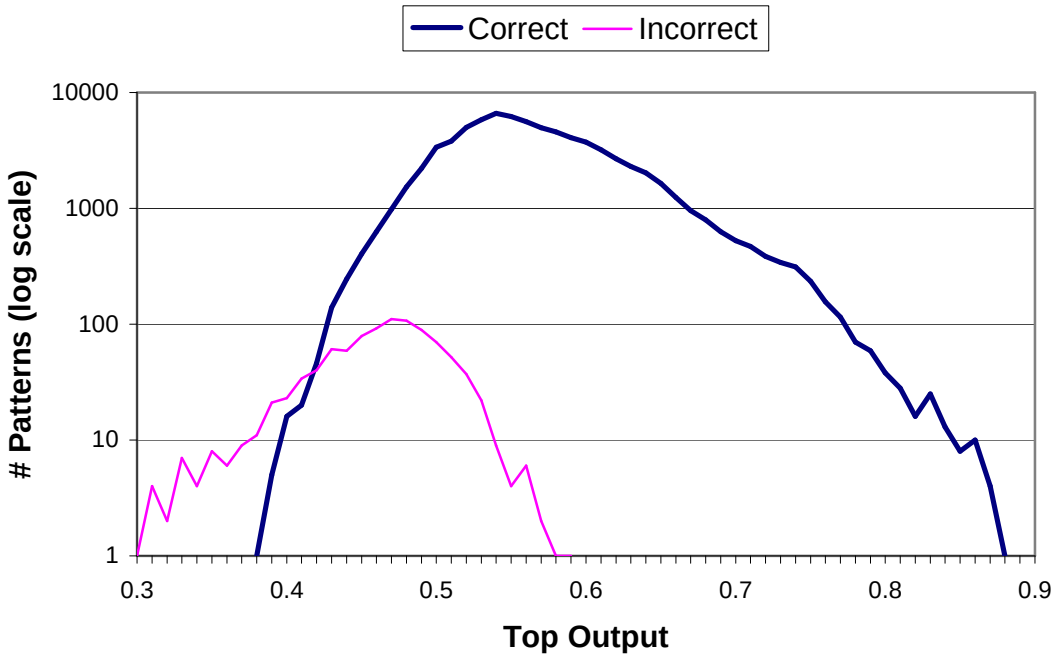

Figure 9. Network outputs on *OCR* test set after CB1 training.

Network weights are typically very small.

5.3 Effect of CB training on output values

CB1 training produces a final output distribution quite unlike that seen in Figure 8. When networks only perform weight updates to prevent misclassification, instead of pushing the pattern outputs to one end of the output range or the other, the vast majority remains spread out just slightly above the decision boundary (see Figure 9). Pattern output distribution is roughly gaussian, reflecting an actual gaussian data distribution (i.e., gaussian noise in the *OCR* input features). There is a larger output variance than appears from SSE optimization but with only a fraction of the classification error. This suggests that the decision surface is much smoother and that network weights are not

saturated. Misclassified patterns usually have outputs below 0.5 and are lower than the output for correctly classified patterns in the majority of cases.

5.4 Network complexity

At first, it seems counter-intuitive that networks outputting only around 0.5 will generalize so well. Ordinarily, training networks together allows a classifier to become more complex, prone to overfitting. However, it has been shown that the number of nodes in a network is not as influential as the *magnitude* of the weights (Bartlett, 1998). The topology, rather, serves more as a mechanism that lends itself to solving of certain problems, while the weights represent how tightly the network has fit itself to the (admittedly incomplete) training data distribution. Network complexity is further defined (Wang, Venkatesh, & Judd, 1994) as the number of parameters and the *capacity to which they are used in learning* (i.e., their magnitude). The authors show how network complexity is a generalization of Akaike's Information Criterion, which reveals

> *The generalization error of a network is affected not only by the number of parameters but also by the degree to which each parameter is actually used in the learning process.*

That is, it is best to make minimal *use* of the capacity of the network for encoding the information provided by the learning patterns (Wang, Venkatesh, & Judd, 1994). In light of this, it is understandable why training (overly complex) networks using early stopping, weight decay or CB training, which allow networks to converge with smaller weights

than normal, perform well. Although a network may have more parameters than strictly necessary, CB1 avoids superfluous weight updates when patterns are correctly classified. This results in lower network complexity. Hence, the possibility of overfitting is reduced in the training process.

The networks used in the *OCR* experiments (1 for each class) had 64 inputs, 32 hidden nodes and 1 output node, with 2080 weight parameters (plus 33 bias weights). The rows of Table 5 list the average magnitude of the weights in a network initialized with uniform random weights in the range [-0.3,0.3], after standard training, and after CB training, respectively. The columns denote the average magnitude of the bias weight on the hidden nodes, bias on the output node, average weight from input to hidden node, and from hidden to output node, respectively. The lowest weight magnitudes are bolded. The CB network has weights that are roughly two to four times larger than the initial random values, while SSE and CE training produce weights from ten to twenty times larger. The CB network is a simpler solution than the networks produced by backpropagation training optimizing SSE or CE.

Table 5. Average final network weight magnitudes.

Method	Hidden Bias	Output Bias	Hidden Weight	Output Weight
Initial	0.16	0.15	0.15	0.15
SSE	2.21	4.66	1.27	6.25
CE	2.56	4.95	1.43	4.16
CB1	**0.56**	**0.02**	**0.31**	**0.74**

5.5 Multi-task learning with CB

Common training methods for learning multiple tasks involve training multiple networks separately, one for each task. However, learning the subparts of a complex problem separately may not be a good idea. Independent training of domain-specific experts is only marginally beneficial to the system as a whole. *Multi-task learning* (MTL), learning multiple problems simultaneously with a single multiple-output network, is described by Caruana (1993; 1995; 1996; 1997). Caruana shows how learning multiple tasks in conjunction helps to avoid local minima and improve generalization. MTL performs better (learning tasks simultaneously) than learning tasks separately (Caruana, 1993).

There are several reasons why MTL improves on single-task learning (STL). Using single-output networks to learn each class in the problem ensures each class is learned separately. Learning classes separately might allow easier analysis of solutions, whereas deciphering the meaning of network weights in a multi-output network is very difficult. However, there are advantages to CB training using a single multi-output network over separate single-output networks. Training a single network takes advantage of the benefits of MTL. Where problem hypotheses overlap, a single network can "reuse" nodes by taking advantage of redundant features. This produces a more compact solution than having to relearn redundant features in separate networks. In experiments on the *OCR* set 47 networks were trained. Each network had a 64x32x1 architecture, (plus bias) yielding 2113 weight parameters in each network. In all, the model contains 2113 x 47 = 99,311 weights, whereas the best single network has a 64x256x47 topology (plus bias). This equals 16640 + 12079 = 28,719 weights, a reduction in size of nearly three-and-a-

half times. The practical implications of this are that not only is memory conserved, but classification speed is increased as well.

Caruana (1995) states one of the disadvantages of MTL is that, since tasks are learned at different times during training, it is difficult to know when to stop training. When training is stopped early, some tasks might not have been learned and generalization is often impaired as a result. Caruana's solution is to train the network until all tasks appear to be overfitting, or to take a separate snapshot of the network for each class, at the point where its validation accuracy is highest. However, taking several snapshots makes the solution much more unwieldy, and although the snapshot is taken at the point where accuracy is highest, there is no guarantee that overfitting has not already occurred in *some part* of the space for that class.

CB training solves both problems by naturally stopping training on tasks as they are learned, both within classes and among them. This helps in two ways: the solution can be kept small (using a single network), and overfitting is discouraged on two levels, both external to learning a class (overfitting a class because other classes have yet to be learned sufficiently) and internal to it (overfitting on localized regions of a class because other regions have yet to be learned).

CB training of multiple networks goes a step beyond MTL. Note that in Section 4.3.1, the best *OCR* test accuracy was obtained using multiple networks. It appears their increased computational ability (more network parameters) over the monolithic model

enables them to be used as needed to find a better solution than with a single multi-output network, while CB training discourages abuse of the increased potential of the system to overfit. In addition to having specialized networks learning individual tasks at the same time, CB explicitly shares relevant information among the networks, in the form of output values, during training to coordinate their learning process.

5.6 Computational Cost

CB1 requires an O(n) search through the n network outputs to determine the highest target and competitor values. However, this additional overhead to the learning algorithm is negligible compared to the computation requirements of O(ih) for feed-forwarding a pattern vector and O(ihn) for backpropagation, where i is the number of inputs and h is the number of hidden nodes. In fact, CB1 saves O(ihn) time by omitting the error backpropagation step over correctly classified patterns. The number of epochs required to converge is similar for CB1 and conventional backpropagation training.

6 Considerations in Neural Network Training using CB1

In this section, several issues are enumerated that must be considered when designing an effective neural network backpropagation learner. How each of these issues is dealt with, to some extent, has a significant effect on generalization. How CB training deals with these issues is discussed.

6.1 Network Complexity

If it is possible to reduce network complexity without reducing training error, then it is expected that generalization accuracy will improve. Network complexity is defined (Wang, Venkatesh, & Judd, 1994) as the number of parameters and the capacity to which they are used in learning (i.e., their magnitude). A network with a few large weights may effectively be more complex than a network with a greater number of small weights. Hence, complexity can be reduced not only through pruning parameters, but also by reducing their values. A learning algorithm that aims at preserving small weights during training can aid in improving generalization. One example of this is performing regularization such as weight decay (Werbos, 1988), which serves to weaken overly strong or saturated connections and in effect remove unused network connections. However, weight decay serves more as a recovery technique to repair the damage caused by minimizing the error function as weights tend toward saturation, rather than providing a heuristic that specifically aims at small-weight solutions. CB1 actively attempts to find good solutions with weights remaining as small as possible to avoid saturation.

6.2 Early Stopping

Early stopping strategies (Wang, Venkatesh, & Judd, 1994) commonly utilize network architectures that have the potential of being overly complex. Larger network architectures are likely to converge to a lower training error, but tend to produce higher error on test patterns. In order to avoid this, early stopping strategies try to determine when the problem has been learned sufficiently well, but not yet overfit (Caruana, 2000).

(Wang, Venkatesh, & Judd, 1994) shows that stopping learning before the global error minimum has the effect of network size selection. This can be accomplished through a number of methods, such as considering the accuracy of a validation, or holdout, set, and stopping training when the performance on the holdout set begins to degrade (Andersen & Martinez, 2001).

CB training performs an "online" form of early stopping. Rather than stopping training completely when it is detected that the training set is being overfit, CB1 selectively omits training on *individual patterns* when backpropagating an error signal would not increase accuracy further.

6.3 Model Complexity

It is often believed that networks with too many degrees of freedom generalize poorly. This line of reasoning is based on two observations: (1) that a sufficiently large network is able to memorize the training data if training continues long enough, and (2) even with early stopping approaches, it is not apparent whether some form of overfit has occurred. By reducing the learning capacity of such a network, it is thereby forced to generalize as it no longer has the capability to memorize the training data.

Caruana (1997, 2000) points out that in order to perform a proper theoretical analysis of network capacity and generalization, the search heuristic must also be taken into account. Gradient descent search heuristics do not give all hypotheses an equal opportunity. The inductive bias of standard backpropagation is to start with a simple hypothesis (through

usually small, random weights) and make the hypothesis more complex (by increasing the magnitude of the weights) until the network sufficiently learns the problem.

Thus, backpropagation is biased toward hypotheses with small weights, examining solutions with larger weights only as dictated by necessity. Excess network capacity does not necessarily hinder generalization, as learning stops as soon as possible. This stopping point is dictated in part by the objective function. During the first part of training, large networks behave like small networks. If they do not come to a satisfactory solution, they begin to perform less like small networks and more like mid-size networks, and so on. If a large network is too big, early stopping procedures will detect when generalization begins to degrade and halt training. At this point, the larger network performs similar to some smaller network. This means that generalization can be less sensitive to excess network capacity, and that using a network that is too small can hurt generalization more than using networks that are too large (Caruana, 1997).

The ability to perform online per-pattern stopping, combinable with standard early stopping techniques, enables CB training to be more robust in its management of excessively large networks. In empirical tests for optimal network sizes in the experiments above, CB1 proved to be more robust to overly large numbers of hidden nodes than SSE and CE optimization.[2]

[2] See Chapter 6, Section 4.4 for published results of an expanded test.

6.4 Overfitting

In taking all of the above issues into account, overfit is typically considered to be a global phenomenon. However, the degree of overfit can vary significantly throughout the input space. (Caruana, Lawrence and Giles, 2000) show that overly complex MLP models can improve the approximation in regions of underfitting, while not significantly overfitting in other regions. However, their discussion is limited to function approximation tasks and not classification problems, which are affected in a different way by bias-variance tradeoffs (Friedman, 1997). CB training seeks to minimize overfit not only globally but also locally by not training on patterns that are already correctly classified.

A network's *bias* and *variance*, as defined in (Geman & Bienenstock, 1992), can be intuitively characterized as the network's test set generalization and its sensitivity to training data, respectively. There exists an inherent tradeoff between bias and variance, namely

> *The best generalization requires a compromise between the conflicting requirements of small variance and small bias. It is a tradeoff between fitting the training data too closely (high variance) and taking no notice of it at all (high bias)* (Sharkey, 1996).

Bias is the extent to which the network's output varies from the target function (the error), while variance is the sensitivity to the training data sampled in affecting

generalization (the variance of the constructed hypothesis from the optimal Bayes hypothesis). An ideal function approximation network has low bias and low variance.

Friedman illustrates that low SSE bias is not important for classification, and one can reduce classification error toward the minimal (Bayes) value by reducing variance alone (Friedman, 1997). One way to reduce variance is by constructing a smoother decision surface. CB1 accomplishes this by discouraging patterns from affecting the shape and location of the decision surface more than is required for correct classification. SSE bias is acceptably increased, as CB training is used for classification tasks, not function approximation.

6.5 Coordinating objective function

CB training provides coordination among multiple single-output networks, or among output nodes in a multi-output network. CB training illustrates the principle of *satisficing* (Simon, 1959), where an aspiration level is specified, such that once that level is met, the corresponding solution is deemed adequate. CB training balances an output's *credibility*, or the exactness with which it can produce ideal target values for its class (*e.g.*, reducing SSE to zero), against its *rejectability*, or the risk of overfitting by doing so. A trade-off is created between exactness in individual class outputs and the classification accuracy of the system. An output node can satisfactorily perform less "ideally" with the understanding that the effectiveness of the entire system can be improved as a result. In relaxing the constraint of optimal credibility, resultant rejectability is reduced.

7 Future Work

There are several directions that future research on CB training will take. CB training variants will be considered for batch and mini-batch learning. The effect of modifying the error margin, μ, in time and in space will be considered. Dynamically updating the value of the error margin as training progresses is a straightforward extension to be evaluated. Softprop, a learning approach combining CB1 and SSE optimization during training by means of the error margin, has shown improvement over CB1 in a preliminary study (Rimer & Martinez, 2004) and a thorough analysis will be presented in future work. Using a value for the error margin local to each training instance and intelligently updating these values as training progresses also shows promise. Also, it has been observed that classification errors between SSE and CB trained networks are highly uncorrelated. Ensembles combining SSE and CB trained networks will be analyzed with the expectation that this will further reduce test error.

8 Conclusion

CB training with the CB1 error function produces less overfit in gradient descent backpropagation training than optimizing SSE and CE. It produces simpler hypotheses than SSE and CE, increasing the probability of better generalization. Its robustness and superior generalization over SSE and CE backpropagation has been demonstrated on several applications. On UCI MLR problems, there was an average increase in accuracy

from 90.7% for optimized SSE networks to 92.1% for CB training performing 10-fold stratified cross-validation. Similarly, there was an increase in test accuracy from 97.86% to 99.11% on a very large OCR data set.

References

Andersen, T. & Martinez, T. (2001). DMP3: A Dynamic Multilayer Perceptron Construction Algorithm. *International Journal of Neural Systems, 11:2,* 145-165.

Barnard, E. (1991). Performance and Generalization of the Classification Figure of Merit Criterion Function. *IEEE Transactions on Neural Networks, 2:2*, 322-325.

Bartlett, P. (1998). The Sample Complexity of Pattern Classification with Neural Networks: The Size of the Weights is More Important than the Size of the Network. *IEEE Trans. Inf. Theory, 44:2*, 525-536.

Bianchini, M., Frasconi, P., Gori, M., and Maggini, M. (1998). Optimal learning in artificial neural networks: A theoretical view. In C. Leondes, (Ed.), *Neural Network Systems Techniques and Applications*, 1-51. Academic-Press.

Blake, C. & Merz, C. (1998). UCI Machine Learning Repository, http://www.ics.uci.edu/~mlearn/MLRepository.html. Irvine, CA: University of California, Department of Information and Computer Science.

Caruana, R. (1993). Multitask Connectionist Learning. Proceedings of the *1993 Connectionist Models Summer School*, (pp. 372-379).

Caruana, R. (1995). Learning Many Related Tasks at the Same Time With Backpropagation. *Advances in Neural Information Processing Systems 7* (Proceedings of NIPS'94), (pp. 657-664).

Caruana, R., Baluja, S. & Mitchell, T. (1996). Using the Future to 'Sort Out' the Present: Rankprop and Multitask Learning for Medical Risk Evaluation. *Advances in Neural Information Processing Systems 8* (Proceedings of NIPS'95).

Caruana, R. (1997). *Multitask Learning*. Ph.D. Thesis, School of Computer Science, CMU.

Caruana, R., Lawrence, S. & Giles, C. (2000). Overfitting in Neural Networks: Backpropagation, Conjugate Gradient, and Early Stopping. Proceedings of *Neural Information Processing Systems, 13*. MIT Press, 2001.

Chakraborty, G., Sawada, M., & Noguchi, S. (1997). Combining Local Representative Networks to Improve Learning in Complex Nonlinear Learning Systems. *IEICE Trans. Fundamentals, E80-A, 9*, 1630-33.

Duda, R., Hart, P. & Stork, D. (2001). *Pattern Classification*, 2nd edition. Wiley, John & Sons, Inc.

Friedman, J. (1997). On Bias, Variance, 0/1-Loss, and the Curse-of-Dimensionality. *Data Mining and Knowledge Discovery, 1:1*, 55-77. Kluwer Academic Publishers.

Geman, S. & Bienenstock, E. (1992). Neural Networks and the Bias/Variance Dilemma. *Neural Computation, 4*, 1-58.

Hampshire II, J. (1990). A Novel Objective Function for Improved Phoneme Recognition Using Time-Delay Neural Networks. *IEEE Transactions on Neural Networks, 1*:*2*.

Jacobs, R., Jordan, M., Nowlan, S., & Hinton, G. (1991). Adaptive Mixtures of Local Experts. *Neural Computation, 3*, 79-87.

Joost, M. & Schiffmann, W. (1998). Speeding up Backpropagation Algorithms by using Cross--Entropy combined with Pattern Normalization. *International Journal of Uncertainity, Fuzziness and Knowledge-based Systems (IJUFKS), 6*:*2*, 117-126.

LeBlanc, M. & Tibshirani, R. (1993). Combining estimates in regression and classification. *NeuroProse*.

LeCun, Y., Denker, J. & Solla, S. (1990). Optimal Brain Damage. In Touretzky, D. (ed.), *Advances in Neural Information Processing Systems (NIPS) 2*, 598-605. San Mateo, CA: Morgan Kaufmann.

Mitchell, T. (1997). *Machine Learning.* McGraw-Hill Companies, Inc., Boston.

Rimer, M., Andersen, T., & Martinez, T. (2001a). Improving Backpropagation Ensembles through Lazy Training. Proceedings of the *IEEE International Joint Conference on Neural Networks IJCNN'01* (pp. 2007-2012).

Rimer, M. & Martinez, T. (2004). Softprop: Softmax Neural Network Backpropagation Learning. Proceedings of the *IEEE International Joint Conference on Neural Networks IJCNN'04* (pp. 979-984).

Rumelhart, David E., Hinton, Geoffrey E. and Williams, Ronald J. (1985). Learning Internal Representations by Error Propagation. Institute for Cognitive Science, University of California, San Diego; La Jolla, CA.

Schiffmann, W., Joost, M., & Werner, R. (1993). Comparison of Optimized Backpropagation Algorithms. *Artificial Neural Networks.* European Symposium, Brussels.

Sharkey, A. (1996). On Combining Artificial Neural Nets. *Connection Science, 8*:*3-4*, 299-313.

Simon, H. (1959). Theories of decision-making in economics and behavioral science. *American Economic Review, XLIX*, 253.

Wang, C., Venkatesh, S., & Judd, J. (1994). Optimal stopping and effective machine complexity in learning. In Cowan, J., Tesauro, G., & Alspector, J. (eds.), *Advances in Neural Information Processing Systems, 6* (pp. 303-310). Morgan Kaufmann, San Francisco.

Werbos, P. (1988). Backpropagation: Past and future. *Proceedings of the IEEE International Conference on Neural Networks* (pp. 343-353). IEEE Press.

Zarndt, F. (1995). A Comprehensive Case Study: An Examination of Machine Learning and Connectionist Algorithms. Masters Thesis, Brigham Young University.

Chapter 3

Improving Speech Recognition Learning through Lazy Training

Michael E. Rimer
Brigham Young University
Computer Science Department
Provo, UT 84602, USA
mrimer@axon.cs.byu.edu

Tony R. Martinez
Brigham Young University
Computer Science Department
Provo, UT 84602, USA
martinez@cs.byu.edu

D. Randall Wilson
Fonix Corporation
180 West Election Road
Draper, Utah 84020
randy@axon.cs.byu.edu

Abstract. Multi-layer backpropagation, like most learning algorithms that can create complex decision surfaces, is prone to overfitting. We present a novel approach, called *lazy training*, for reducing the overfit in multiple-layer networks. Lazy training consistently reduces generalization error of optimized neural networks by more than half on a large OCR dataset and on several real world problems from the UCI machine learning database repository. Here, lazy training is also shown to be effective in a multi-layered adaptive learning system, reducing the error of an optimized backpropagation network in a speech recognition system by 50.0% on the TIDIGITS corpus.

1 Introduction

Multi-layer feed-forward neural networks trained through backpropagation have received substantial attention as robust learning models for tasks including classification [17]. Much research has gone into improving their ability to generalize beyond the training data. Many factors play a role in their ability to learn, including network topology,

learning algorithm, and the nature of the problem being learned. Overfitting the training data, caused through the use of an inappropriate objective function, is often detrimental to generalization. In applications such as speech recognition where even a small amount of error can be unacceptable it is important to generalize as well as possible.

This work introduces *word training* (WT), a novel technique for training speech recognition networks. Word training, inspired by *lazy training* [15], implements an objective function that seeks to directly minimize word classification error while discouraging overfitting. Lazy training performs successfully on a large OCR dataset and several problems selected from the UCI machine learning database repository, reducing their average generalization error over training of optimized networks by more than 60% using 10-fold cross-validation [17]. An extensively optimized, state-of-the-art backpropagation network achieves word recognition error of 0.12% on the TIDIGITS speech recognition corpus [11]. Word training performs markedly better than optimized standard backpropagation training, decreasing test set error by half, from 0.12% to 0.06%.

An overview of related work and a discussion of objective functions are provided in Section 2. The lazy training and the word training algorithms are presented in Section 3. Experiments and results are given in Section 4. Analysis and discussion are in Section 5. Conclusions and future work are presented in Section 6.

2 Related work

The speech recognition problem is very complex and has received much attention in machine learning literature. Many learning models have been developed to cope with the difficulty of this problem. Often, neural networks have been utilized to provide a solution. However, neural networks are prone to overfit to the training data, which is detrimental to robust generalization. *Hidden Markov models* (HMMs) traditionally perform as well or better than neural networks at speech recognition [14]. Word training achieves results comparable to HMMs.

2.1 Critique of current training techniques

To generalize well, a learner must have a proper objective function. Most learning techniques incorporate an objective function of minimizing *sum-squared-error* (SSE). The validity of using SSE as an objective function to minimize error relies on the assumption that sample outputs are offset by inherent gaussian noise, being normally distributed about a cluster mean. For learning function approximation of an arbitrary signal, this presumption often holds. However, this assumption is invalid for classification problems, where the target vectors are class codings (i.e., arbitrary nominal or boolean values representing designated classes).

Cross-entropy (CE) assumes *idealized* class outputs (i.e., target values of zero or one for a sigmoid activation) [13] and is therefore more appropriate to classification problems. However, error values using SSE and cross-entropy have been shown [9] to be

inconsistent with ultimate sample classification accuracy. That is, minimizing CE or SSE is not necessarily correlated to high recognition rates. Numerous experiments in the literature provide examples of networks that achieve little error on the training set but fail to achieve the best possible accuracy on test data [2, 18]. This is due to a variety of reasons, such as *overfitting* the data or having an incomplete representation of the data distribution in the training set. There is an inherent tradeoff between fitting the (limited) data sample perfectly and generalizing accurately over the entire population.

2.2 Shortcomings of search methodologies

More fundamentally, the above objective functions provide mechanisms that do not reflect the true goal of classification learning, which is to achieve high recognition rates on unseen data. In [9], a monotonic objective function, the *classification figure-of-merit* (CFM), is introduced for which minimizing error remains consistent with increasing classification accuracy. Networks that use the CFM as their criterion function in phoneme recognition are introduced in [9] and further considered in [5]. They are, however, also susceptible to overfitting. The question of how to prevent overfitting is a subtle one. When a network has many free parameters, learning is fast and also local minima can often be avoided. On the other hand, networks with few free parameters tend to exhibit better generalization performance. Determining the appropriate size network remains an open problem [8].

The problem of overfitting has received much attention in the literature. Methods of addressing this problem include using a holdout set to stop training early [20], cross-

validation [2], node pruning [7, 8], and weight decay [21], among others. These techniques approach optimal solutions given the bias of standard backpropagation learning but do not consider possible enhancements to the bias itself. Node pruning seeks to improve accuracy by simplifying network topology, rather than alleviating the problems common to larger topologies, for example. Methods for overcoming problems in the inductive bias inherent to training with backpropagation generally involve forming network ensembles. Ensemble techniques, such as *bagging* and *boosting* [12], or *wagging* [3], are more robust than single networks when the errors among the networks are not positively correlated.

There is evidence that the size of the weights in a network plays a more important role to generalization than the number of nodes [4]. A simple method of reducing overfitting is to provide a maximum error tolerance threshold, d_{max}, which is the smallest absolute output error to be backpropagated. In other words, no weight update occurs for a given d_{max}, target value, t_k, and network output, o_k, if the absolute error $| t_k - o_k | < d_{max}$. This threshold is arbitrarily chosen to represent a point at which a sample has been sufficiently approximated. With an error threshold, the network is permitted to converge with much smaller weights [19].

3 Word training method

This work addresses overfitting exhibited by previous backpropagation solutions by applying *lazy training*, a conservative form of training, to the learning process (see

Section 3.3). Similar to CFM, it requires that a reduction in error correlate to increasing accuracy. However, CFM does not prevent weight saturation, which is often detrimental to accuracy [4]. Lazy training only backpropagates an error signal from output nodes that endanger classification accuracy. This approach allows the model to approach a solution more conservatively and discourages overfitting.

3.1 Phoneme training algorithm

Speech recognition is a complex problem, and a standard approach involves simplifying the problem by breaking it up into smaller, simpler ones. Word recognition is broken into the simpler problem of phoneme recognition. The signal is divided into small time slices called *frames* and features derived from each frame are input into the recognizer (see Figure 1). The recognizer then outputs the probability of each phoneme being uttered during that frame. Often, several contiguous frames are considered simultaneously, as in the multi-layer time-delay neural network in [10]. Phonemes are identified and combined through a proper linguistic model to derive words. However, derived features of a speech signal are often noisy and speaker dependent. Hence, it is difficult to achieve a satisfactorily high phoneme recognition rate at each frame and produce a reasonable solution.

Therefore, a decoder is stacked onto the phoneme recognizer to provide a more holistic solution. The decoder receives the outputs of the phoneme recognizer and combines the outputs over time to make a more educated guess as to what word or phrase has been spoken. Pairings of adjacent possible phonemes are validated or prohibited according to

the linguistic model, and the overall most-likely sequence of phonemes is output as the response. Additional elements such as a lexicon can be incorporated into the decoder to constrain possible responses to produce more intelligent solutions. The decoder can be made even more sophisticated to combine probable words together into entire utterances according to a language model.

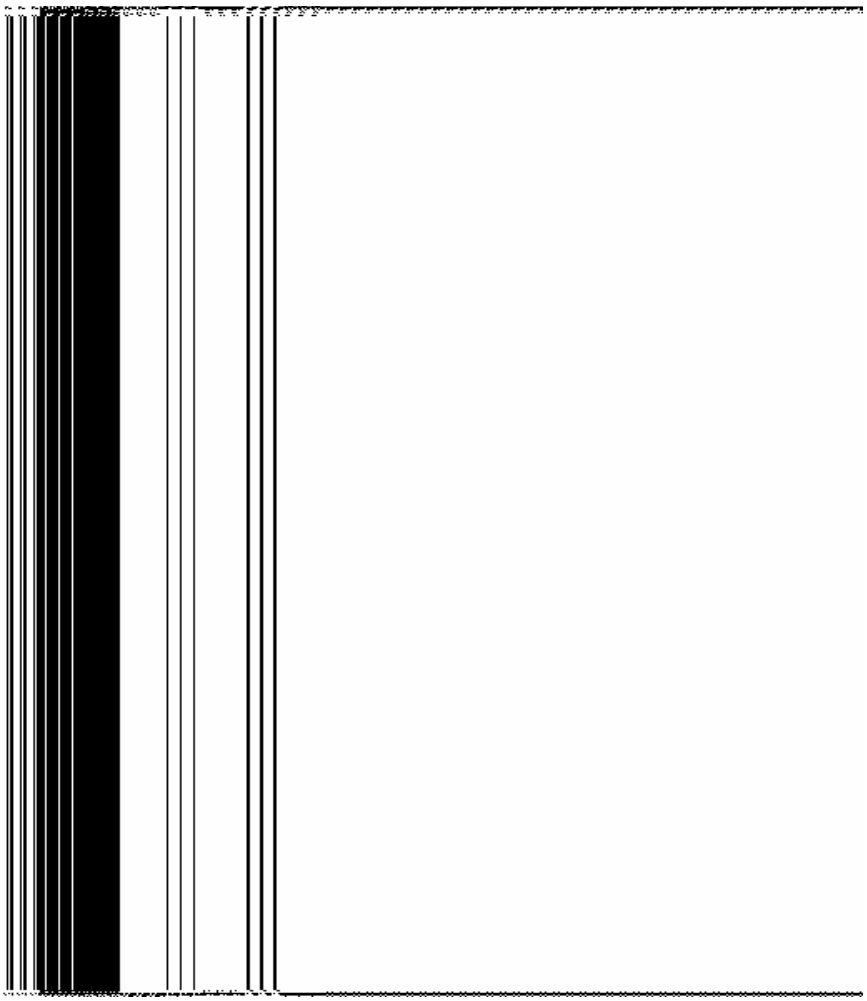

Figure 1. Word training system with neural network and decoder.

Phoneme training involves presenting a series of utterances to the network. Each utterance is divided into temporal frames and features derived from the signal that are input into the network. Each frame is labeled with the phoneme being spoken during that

time. The network is often trained using backpropagation with a cross-entropy objective function.

3.2 Lazy training paradigm

Due to the reasons stated in Section 2, a neural network classifier often overfits the training data. The tendency to overfit is further aggravated because labeled data points in this problem space are sparse. The problem is compounded since phonemes blend together, and it is problematic to label minute time slices accurately. It is therefore desirable to incorporate a recognizer that will overfit as little as possible in order to produce the highest possible generalization accuracy.

Overfitting a neural network is often equated with saturating the weights. It follows that overfit is reduced by letting the weights be as small as possible in the solution. This ideal can be approached through the following method.

For each frame considered by the recognizer during training, only those outputs that are credited with classification errors are updated through backpropagation. The result is training without idealized target outputs of 0 and 1, providing a learning mechanism that is reminiscent of constraint satisfaction and reinforcement learning, where the network outputs learn to interact with their (changing) environment (the behavior of the decoder based on the values of the output nodes). As this forces networks to learn only when explicit evidence is presented that their state is a detriment to classification accuracy, we have dubbed this technique *lazy training* (not to be confused with *lazy learning*

approaches [1]). Backpropagation training often uses an objective function that tends to a *saturation* of the weights. That is, it tends to encourage larger weights in an attempt to output a value approaching the limits of 1 or 0. The ramifications of this are discussed further in Section 5. Lazy training is biased toward simpler solutions, meaning that smaller weights (even approaching zero) can be used to provide an acceptable solution.

Two or more output nodes can in effect collaborate together to decide how learning is to proceed at any given point. More specifically, interaction among outputs allows a *dynamic error threshold* to be implemented. That is, when one output presents a sufficient solution in an area of the problem space, other outputs do not need to work at redundantly modeling the same local data. Consequently, they are able to specialize and break a complex problem up into smaller, simpler ones. This provides for a more conservative form of training that converges with smaller network weights, hence with less overfitting and greater generalization accuracy.

The lazy training methodology has been successfully utilized to significantly reduce error on OCR data and on several problems from the UCI repository of machine learning databases [6,15]. We implement it here for speech recognition to show further advantages of this training style. In past experiments, lazy training was performed on N separate single-output networks (one for each class in the problem). Here we show how it can successfully be used on a single N-output network. A single network provides a more compact, simpler, faster solution than many separate networks in learning a problem with several output classes.

Also, we illustrate that lazy training learns effectively when there is a level of indirection necessarily involved in arriving at a solution. In this case, while the network learns to output phoneme confidences, these confidences do not provide the actual solution, but are used by the phoneme decoder to derive the words spoken. High phoneme accuracy is therefore not necessarily the goal of training, but instead high word recognition rates. Word training (WT) is the name we give to training with an objective of directly increasing word recognition accuracy (possibly at the expense of phoneme accuracy). The method for deducing the network phoneme error from word error is presented in the following sub-section.

3.3 Word training algorithm

In word training the network decoder is involved in the training process. The decoder gathers the network outputs on all the frames of an utterance. When the decoder outputs a recognized word sequence, the output is compared against the target word sequence. If the output utterance matches the target, no error signal is propagated through the network at all (see Fig. 1, Error Signal). The network performs adequately within the system, and refraining from updating the weights discourages overfitting. When a discrepancy between the output and target exists, then the network weights are updated only on those time frames where the word errors occur.

Let N be the number of network output nodes (distinct class labels). Let o_k be the output value of the k^{th} output node of the network ($0 \leq o \leq 1$, $1 \leq k \leq N$). Let T designate the

target output class for a given frame and c_k signify the class label of the k^{th} output node. For target output nodes, $c_k = T$, and for non-target output nodes, $c_k \neq T$. Non-target output nodes are called *competitors*. Let $o_{T\max}$ denote the highest-outputting target output node. Let $o_{C\max}$ denote the value of the highest-outputting competitor. The error, ε_k, back-propagated from the k^{th} output node of the network is defined as

$$\varepsilon_k \equiv \begin{cases} \tau_U - o_k & \text{if } c_k = T \text{ and } (o_{C\max} \geq o_{T\max}) \\ \tau_L - o_k & \text{if } c_k \neq T \text{ and } (o_k \geq o_{T\max}) \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where τ_U and τ_L are upper and lower target values such that $0 \leq \tau_L < o_k < \tau_U \leq 1$. Thus, the target output generates an error signal only if there is some competitor with an equal or higher value than $o_{T\max}$, signaling a potential misclassification. Non-target outputs generate an error signal only if they have an output equal to or higher than $o_{T\max}$, indicating they are responsible for the misclassification.

The rate of convergence is partly dependent on the values used for τ_U and τ_L. Note that changing either τ is effectually equivalent to altering the learning rate. A τ closer to the current output value o_k implies a smaller error signal and will result in slower, but steadier convergence that more closely approximates the true error gradient than values near 0 or 1.

Word training of a network proceeds at a different pace than with standard backpropagation phoneme training. Training only the nodes that directly contribute to classification error of a word allows the model to relax more gradually into a solution,

learning only as much as it needs to and thereby discouraging overfitting. This approach is reminiscent of training with an error threshold; however whereas a fixed error threshold causes training to stop at a pre-specified point, word training dynamically halts at the first possible point for a given frame at a given point in time. Weights are updated only through necessity. Without the decoder, a phoneme can be considered "learned" with any output value, providing competitors output lower values. Using a decoder, even more flexibility is possible, since the target output on a phoneme can be lower than its competitors and a word still be correctly identified.

Additionally, overfitting is minimized in a word trained network because outliers (noisy frames) have minimal detrimental impact to the decision surface's accuracy. This is because the target output is only required to output a value that is negligibly higher than the output representing the neighboring class, as illustrated in Figure 2b. This is in contrast to classical gradient descent training, where hard target values of 0 and 1 are required (translating to pushing the decision surface as far away as possible) even for outliers as illustrated in Figure 2a. Hence, in testing, samples close to the outlier belonging to the competing class (represented by the question mark) have a much better chance of being correctly classified.

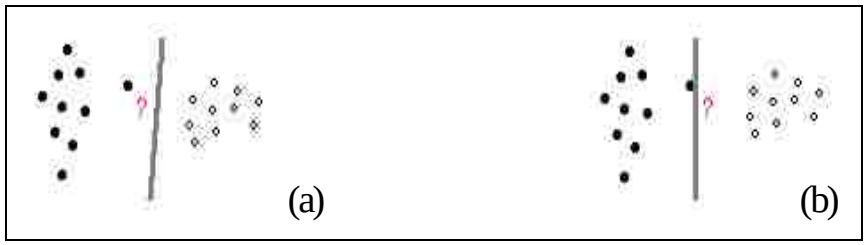

Figure 2: Overfit decision surface (a) and lazy-trained surface (b).

3.4 Enlarging the margin

When lazy training, it is common for the highest outputting node in the network to output a value only slightly higher than the second-highest-firing node (see Figure 3). This is true for correctly classified samples (above 0 in Figure 3), and also for incorrect ones (below 0). This means that most training samples remain physically close to the decision surface throughout training. An error margin, μ, can be introduced during the training process that serves as a confidence buffer between the outputs of target and competitor nodes. Under the sigmoid function, the error margin is bounded by [–1, 1]. For no error signal to be backpropagated from the target output, an error margin requires that $o_{C\max} < o_{T\max} - \mu$. Conversely, for a competing node k with output o_k, the inequality $o_k < o_{T\max} - \mu$ must be satisfied for no error signal to be backpropagated from k.

During the training process, μ can be increased gradually and might even be negative to begin with, not expressly requiring correct classification at first. This gives the networks time to configure their parameters in an even more uninhibited fashion. Then μ is increased to an interval sufficient to account for the variance that appears in the test data, allowing for robust generalization. The value of μ can also be decreased, and remain negative as training is concluded to account for noisy outliers (see Section 5.1).

At the extreme value of μ equal to 1, lazy training becomes standard SSE training, with target values of 1.0 and 0.0 required for all positive and negative samples, respectively.

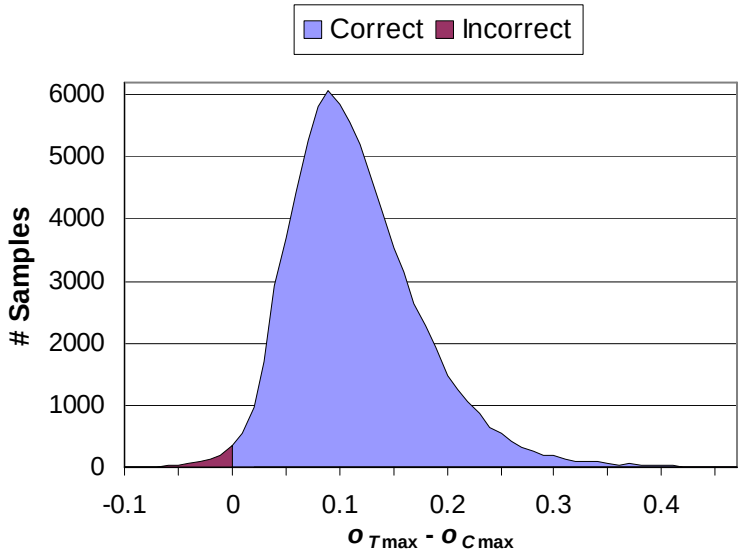

Figure 3: Network output margin of error after lazy training.

4 Experiments

The performance of phoneme versus word training models has been evaluated on a subset of the TIDIGITS data corpus consisting of over 17,000 utterances and sampled at 11 kHz, containing 50,000 spoken digits, partitioned into roughly 15,000 training samples, 1,000 validation samples and 1,000 test samples. Each sample is partitioned into 10 ms frames. The features generated for input to the network are standard mel-cepstral coefficients and their derivatives.

4.1 Parameters

We compared fully connected feed-forward network trained through on-line backpropagation maximizing cross-entropy on single frames against word-trained networks trained on utterances. In the experiments presented, networks contained a

single hidden layer comprised of 50, 100, or 200 hidden nodes. Weights were initialized to small random values. The same initial weights were used for each training method. The learning rate began at 0.05 and a harmonic decay frequency of 5 epochs was used. In these tests a τ_U of 1 and τ_L of 0 were used for faster lazy training, and μ was 0. Training was halted after 150 epochs, many epochs after training error ceased to decrease.

The backpropagation network used is state-of-the-art. Its topology, objective function and learning parameters were optimized through extensive experimentation over a period of several years.

4.2 Results

Table 1 displays the test results of standard CE back-propagation training (BP) versus word training (WT). Accuracies are shown in percent. Highest column values are shown in bold, with the highest value for the other learning technique italicized. Note that high word accuracy is our prime goal. High sentence accuracy is a desired consequence, and phoneme accuracy is ultimately irrelevant.

Table 1. Results on subset of TIDIGITS data corpus.

Method, Hidden Nodes	Phoneme	Base phoneme	Word	Sentence
BP 200	**79.33**	**91.93**	*99.88*	*99.60*
BP 100	74.58	89.48	99.73	99.10
BP 50	66.40	84.66	99.71	99.00
WT 200	*51.50*	*76.04*	**99.94**	**99.80**
WT 100	47.96	74.03	99.82	99.40
WT 50	46.23	72.05	99.79	99.30

5 Analysis and discussion

Table 1 shows that networks generated through word training have the capability of cutting word error in half from 0.12% for standard phoneme backpropagation training to 0.06% for word training. These tests show that, although word training experienced much lower phoneme accuracy, word accuracy was increased and the amount of overfit was reduced (see Section 5.3). The highest accuracies were achieved with a 200-node hidden layer. Larger networks show no further improvement. Interestingly, as smaller hidden layers are used, word and phoneme accuracy degrades more gracefully for word training than for CE training. When the training process concentrates directly on word accuracy instead of on learning phonemes, not directly responsible for word accuracy, more network parameters are free to learn a better solution.

5.1 Lazy training analysis

When networks are lazy-trained, instead of pushing the sample outputs to one end of the output range or the other, the vast majority remains spread out just slightly above the decision boundary. Output distribution is roughly gaussian, reflecting an actual gaussian data distribution, with a larger variance than appears from standard backpropagation, but only a fraction of the classification error. This suggests that the decision surface is much smoother and that network weights are not saturated.

Training set accuracy is largely preserved on the test set. Since the outputs learn together, their solutions are highly correlated and their solution transfers well to unseen

data. Error is 50.0% less than with phoneme-trained networks, presenting a strong case for lazy training on complex data sets where backpropagation networks tend to overfit.

Lazy training also assists in the case of noisy data and inaccurate or uncertain phoneme labeling. In this case, the output representing the more accurate phoneme can fire roughly equal to the falsely labeled phoneme, rather than forcing it all the way down at 0. Even though the correct phoneme does not fire the highest value among the outputs, it fires nearly that high, enabling the decoder to more easily produce the correct answer.

5.2 Network complexity

The network outputs the majority of values at about 0.5. At first, it seems counter-intuitive that networks outputting only around 0.5 will generalize so well. Ordinarily, training networks together allows a classifier to become more complex, prone to overfitting. According to Occam's razor, adding parameters to a network, beyond the smallest correct solution for a given problem, can be a detriment to the generalization ability of the network. This is similar to the claim that a network with higher learning capacity tends to "memorize" noise in the data, which is an undesirable trait.

Recently, however, it has been illustrated how the number of nodes in a network is not as influential as the *magnitude* of the weights [4]. The topology, rather, serves more as a mechanism that lends itself to solving of certain problems, while the weights represent how tightly the network has fit itself to the (admittedly incomplete) training data distribution. Network complexity is further defined in [20] as the number of parameters

and the *capacity to which they are used in learning* (i.e., their magnitude). In light of this, it is understandable why complex networks and lazy training, which allows networks to have small weights, perform so well together. Although the WT network has a high number of parameters, lazy training prevents further weight updates once frames are correctly classified and results in low complexity. Hence, the possibility of overfitting is reduced in the training process.

The networks used in our experiments had 130 inputs, 50, 100, or 200 hidden nodes and 199 output nodes, with 16,450, 32,900, and 65,800 weight parameters, respectively. The rows of Table 2 list the average magnitude of the weights in networks initialized with small random weights, during phoneme training, and during word training, respectively. The particular values shown are taken following the epoch with the highest word accuracy on the holdout set. The columns denote the average weight from input to hidden nodes, and from hidden to output nodes, respectively. The word-trained network has weights that are twice as large as the initial random values, while standard training produces weights four times larger. The lazy-trained network is a simpler solution than the network produced by standard backpropagation training.

Table 2. Average network weights.

Method	Hidden Weights	Output Weights
Initial	.132	.150
Standard	.491	.567
Lazy	.280	.256

6 Conclusion and future work

Word training reduces overfitting in gradient descent backpropagation training, increasing the probability of discovering better solutions. Its advantages in word recognition over standard backpropagation phoneme training have been demonstrated in a speech recognition system. A word-trained network reduces word recognition error by half over an optimized backpropagation network on the TIDIGITS corpus, a large real world application.

For the word training nets presented, the learning parameters of the optimized backpropagation network were used. No attempt was made to optimize them for lazy training. Since standard backpropagation and lazy training vary significantly in their search technique, it would be expected that different parameter values would perform optimally with each objective function. Different settings on parameters such as τ_U, τ_L, and μ will be tested to further improve generalization accuracy. Word training will be applied to other problems that are broken into smaller pieces and then merged together, such as text recognition, using networks for OCR.

Acknowledgments

This research was funded in part by a grant from *fonix* Corp.

References

[1] David W. Aha, editor, *Lazy Learning*, Kluwer Academic Publishers, Dordrecht, May 1997.

[2] Andersen, Tim and Tony R. Martinez, "Cross Validation and MLP Architecture Selection", *Proceedings of the IEEE International Joint Conference on Neural Networks IJCNN'99*, CD Paper #192, 1999.

[3] Andersen, Tim and Martinez, Tony, "Wagging: A learning approach which allows single layer perceptrons to outperform more complex learning algorithms", *Proceedings of the IEEE International Joint Conference on Neural Networks IJCNN'99*, CD Paper #191, 1999.

[4] Bartlett, Peter L., "The Sample Complexity of Pattern Classification with Neural Networks: The Size of the Weights is More Important than the Size of the Network", *IEEE Trans. Inf. Theory*, 44(2), 1998, pp. 525-536.

[5] Barnard, Etienne, "Performance and Generalization of the Classification Figure of Merit Criterion Function", *IEEE Transactions on Neural Networks*, 2(2), March 1991, pp. 322-325.

[6] Blake, C.L. & Merz, C.J. (1998). UCI Repository of machine learning databases, http://www.ics.uci.edu/~mlearn/MLRepository.html. Irvine, CA: University of California, Department of Information and Computer Science.

[7] Castellano, G., A. M. Fanelli and M. Pelillo, "An empirical comparison of node pruning methods for layered feed-forward neural networks", *Proc. IJCNN'93-1993 Int. J. Conf. on Neural Networks*, Nagoya, Japan, 1993, pp. 321-326.

[8] Castellano, G., A. M. Fanelli, and M. Pelillo, "An iterative pruning algorithm for feed-forward neural networks", *IEEE Transactions on Neural Networks*, Vol. 8 (3), 1997, pp. 519-531.

[9] Hampshire II, John B., "A Novel Objective Function for Improved Phoneme Recognition Using Time-Delay Neural Networks", *IEEE Transactions on Neural Networks*, Vol. 1, No. 2, June 1990.

[10] H. Hild and A. Waibel. "Connected Letter Recognition with a Multi-State Time Delay Neural Network." Neural Information Processing Systems (NIPS-5), 1993.

[11] R. Gary Leonard and George Doddington. (1993). TIDIGITS speech corpus, http://morph.ldc.upenn.edu/Catalog/LDC93S10.html. Texas Instruments, Inc.

[12] Maclin, R and Opitz, D, "An empirical evaluation of bagging and boosting", *The Fourteenth National Conference on Artificial Intelligence*, 1997.

[13] Mitchell, Tom, *Machine Learning.* McGraw-Hill Companies, Inc., Boston, 1997.

[14] Rabiner, Lawrence and Juang, Biing-Hwang, *Fundamentals of Speech Recognition.* Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1993.

[15] Rimer, Michael E., Anderson, Timothy L. and Martinez, Tony R., "Improving Backpropagation Ensembles through Lazy Training", Proceedings of the *IEEE International Joint Conference on Neural Networks IJCNN'01*, pp. 2007-2112, 2001.

[16] Rimer, Michael, E., "Lazy Training: Interactive Classification Learning," Masters Thesis, Brigham Young University, 2002.

[17] Rumelhart, David E., Hinton, Geoffrey E. and Williams, Ronald J., "Learning Internal Representations by Error Propagation", Institute for Cognitive Science, University of California, San Diego; La Jolla, CA, 1985.

[18] Schiffmann, W., Joost, M. and Werner, R., "Comparison of Optimized Backpropagation Algorithms", *Artificial Neural Networks*, European Symposium, Brussels, 1993.

[19] Schiffmann, W., Joost, M. and Werner, R., "Optimization of the Backpropagation Algorithm for Training Multilayer Perceptions", University of Koblenz: Institute of Physics, 1994.

[20] Wang, C., Venkatesh, S. S., and Judd, J. S., "Optimal stopping and effective machine complexity in learning", in Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems*, vol. 6, Morgan Kaufmann, San Francisco, 1994, pp. 303-310.

[21] Werbos, P., "Backpropagation: Past and future", *Proceedings of the IEEE International Conference on Neural Networks*, IEEE Press, 1988, pp. 343-353.

Chapter 4

Softprop: Softmax Neural Network Backpropagation Learning

Michael Rimer
Computer Science Department
Brigham Young University
Provo, UT 84602, USA
E-mail: mrimer@axon.cs.byu.edu

Tony Martinez
Computer Science Department
Brigham Young University
Provo, UT 84602, USA
E-mail: martinez@cs.byu.edu

Abstract. Multi-layer backpropagation, like many learning algorithms that can create complex decision surfaces, is prone to overfitting. Softprop is a novel learning approach presented here that is reminiscent of the softmax explore-exploit Q-learning search heuristic. It fits the problem while delaying settling into error minima to achieve better generalization and more robust learning. This is accomplished by blending standard SSE optimization with lazy training, a new objective function well suited to learning classification tasks, to form a more stable learning model. Over several machine learning data sets, softprop reduces classification error by 17.1% and the variance in results by 38.6% over standard SSE minimization.

1 Introduction

Multi-layer feed-forward neural networks trained through backpropagation have received substantial attention as robust learning models for classification tasks [15]. Much research has gone into improving their ability to generalize beyond the training data. Many factors play a role in their ability to learn, including network topology, learning

algorithm, and the nature of the problem at hand. Overfitting the training data is often detrimental to generalization and can be caused through the use of an inappropriate objective function.

Lazy training [12,13] is a new approach to neural network learning motivated by the desire to increase generalization in classification tasks. Lazy training implements an objective function that seeks to directly minimize classification error while discouraging overfitting. Lazy training is founded upon a satisficing philosophy [9] where the traditional goal of optimizing network output precision is relaxed to that of merely selecting hypotheses that produce rational (correct) decisions. Lazy training has been shown to decrease overfitting and discourage weight saturation in complex learning tasks while improving generalization [13,14]. It has performed successfully on speech recognition tasks, a large OCR data set and several benchmark problems selected from the UCI Machine Learning Repository, reducing average generalization error over training of optimized standard backpropagation networks using 10-fold stratified cross-validation.

In this work a method for combining standard backpropagation learning and lazy training is presented that we call *softprop*. It is named after the softmax exploration policy in Q-learning [19], combining greedy exploitation and conservative exploration in an optimization search. This exploration policy tends to be effective in complex problem spaces that have many local minima. This technique is shown to achieve higher accuracy and more robust solutions than either standard backpropagation or lazy training alone.

A background discussion of traditional objective functions and the lazy training objective function is provided in Section 2. The proposed softprop technique is presented in Section 3. Experiments are detailed in Section 4. Results and analysis are shown in Section 5. Conclusions and future work are presented in Section 6.

2 Motivation for Lazy Training

To generalize well, a learner must use a proper objective function. Many learning techniques incorporate an objective function minimizing *sum-squared-error* (SSE). The validity of using SSE as an objective function to minimize error relies on the assumption that sample outputs are offset by inherent gaussian noise, being normally distributed about a cluster mean. For function approximation of an arbitrary signal, this presumption often holds. However, this assumption is invalid for classification problems where the target vectors are class codings (i.e., arbitrary nominal or boolean values representing designated classes).

Error optimization using SSE as the measure has been shown [8] to be inconsistent with ultimate sample classification accuracy. That is, minimizing SSE is not necessarily correlated to achieving high recognition rates. In [8], a monotonic objective function, the *classification figure-of-merit* (CFM), is introduced for which minimizing error remains consistent with increasing classification accuracy. Networks that use the CFM as their criterion function in phoneme recognition are introduced in [8] and further considered in [5]. They are, however, also susceptible to overfitting.

The question of how to prevent overfitting is a subtle one. When a network has many free parameters local minima can often be avoided. On the other hand, networks with few free parameters tend to exhibit better generalization performance. Determining the appropriate size network remains an open problem [7].

The above objective functions provide mechanisms that do not directly reflect the ultimate goal of classification learning, i.e., to achieve high recognition rates on unseen data. Numerous experiments in the literature provide examples of networks that achieve little error on the training set but fail to achieve high accuracy on test data [2, 16]. This is due to a variety of reasons, such as *overfitting* the data or having an incomplete representation of the data distribution in the training set. There is an inherent tradeoff between fitting the (limited) data sample perfectly and generalizing accurately over the entire population.

Methods of addressing overfit include using a holdout set for model selection [18], cross-validation [2], node pruning [6, 7], and weight decay [20]. These techniques seek to compensate for the bias of standard backpropagation learning [11] in specific situations. For example, as overly large networks tend to overfit, node pruning seeks to improve accuracy by simplifying network topology. Forming network ensembles can also reduce problems in the inductive bias inherent to gradient descent. Ensemble techniques, such as *bagging* and *boosting* [10], or *wagging* [3], are more robust than single networks when the errors among the networks are not closely correlated.

There is evidence that the magnitude of the weights in a network plays a more important role to generalization than the number of nodes [4]. Optimizing SSE tends to a saturation of weights, often equated with overfitting. It follows that overfit might be reduced by keeping the weights smaller. Weight decay is a common technique to discourage weight saturation. Another simple method of reducing overfit is to provide a maximum error tolerance threshold, d_{max}, which is the smallest absolute output error to be backpropagated. In other words, for a given d_{max}, target value, t_k, and network output, o_k, no weight update occurs if the absolute error $| t_k - o_k | < d_{max}$. This threshold is arbitrarily chosen to indicate the point at which a sample has been sufficiently approximated. Using an error threshold, a network is permitted to converge with much smaller weights [17].

2.1 Lazy Training

Retaining smaller weights can be accomplished more naturally through lazy training. Lazy training only backpropagates an error signal on misclassified patterns. Previous work [12, 13] has shown how applying lazy training to classification problems can consistently improve generalization.

For each pattern considered by the network during the training process, only output nodes credited with classification errors backpropagate an error signal. As this forces a network to delay learning until explicit evidence is presented that its state is a detriment to classification accuracy, we have dubbed this technique *lazy training* (not to be confused with *lazy learning* approaches [1]). Often, an objective function is used in

backpropagation training that tends to a saturation of the weights. That is, it tends to encourage larger weights in an attempt to output values approaching the limits of 0 and 1. Lazy training does not depend on idealized target outputs of 0 and 1. As such, it is biased toward simpler solutions, meaning that smaller weight magnitudes (even approaching zero) can provide a solution with high classification accuracy. This approach allows the model to approach a solution more conservatively and discourages overfit.

2.2 Lazy Training Heuristic

The lazy training error function is as follows. Let N be the number of network output nodes (distinct class labels). Let o_k be the output value of the k^{th} output node of the network ($0 \le o \le 1$, $1 \le k \le N$) for a given pattern. Let T designate the target output class for that pattern and c_k signify the class label of the k^{th} output node. For target output nodes, $c_k = T$, and for non-target output nodes, $c_k \neq T$. Non-target output nodes are called *competitors*. Let $o_{T\max}$ denote the highest-outputting target output node. Let $o_{\sim T\max}$ denote the value of the highest-outputting competitor. The error, ε_k, back-propagated from the k^{th} output node of the network is defined as

$$\varepsilon_k \equiv \begin{cases} o_{\sim T\,\max} - o_k & \text{if } c_k = T \text{ and } (o_{\sim T\,\max} \ge o_{T\,\max}) \\ o_{T\,\max} - o_k & \text{if } c_k \neq T \text{ and } (o_k \ge o_{T\,\max}) \\ 0 & \text{otherwise} \end{cases} . \qquad (1)$$

Thus, the target output backpropagates an error signal only if there is some competitor with an equal or higher value than it, signaling a misclassification. Non-target outputs generate an error signal only if they have a value equal to or higher than $o_{T\max}$, indicating

they are also responsible for the misclassification. The error value is set to the difference in value between the target and competitor nodes.

Lazy training of a network proceeds at a different pace than with standard SSE minimization. Weights are updated only through necessity. Hence, a pattern can be considered "learned" with any combination of output values, providing competitors output lower values than targets. Training only nodes that directly contribute to classification error allows the model to relax more gradually into a solution and avoid premature weight saturation.

The output nodes can in effect collaborate together to form correct decisions. When the target output node presents a sufficient solution value in a local area of the problem space (i.e. its value is higher than for non-target nodes), competitor outputs do not need to work at redundantly modeling the same local data (i.e., approximate a zero output value). Consequently, they are able to specialize and break complex problems up into smaller, simpler ones. Whereas a fixed error threshold causes training to stop when output values reach a pre-specified point (e.g. 0.1 and 0.9), lazy training implements a *dynamic error threshold*, halting training on a given pattern as soon as it is classified correctly. Keeping weights smaller allows for training with less overfit and greater generalization accuracy.

2.3 Adding an error margin to lazy answers

When lazy training, it is common for the highest outputting node in the network to output a value only slightly higher than the second-highest-firing node (see Figure 1). This is

true for correctly classified samples (to the right of 0 in Figure 1), and also for incorrect ones (to the left of 0). This means that most training samples remain physically close to the decision surface throughout training. An error margin, μ, is introduced during the training process to serve as a confidence buffer between the outputs of target and competitor nodes. Using the sigmoid function, the error margin is bounded by [–1, 1]. For no error signal to be backpropagated from the target output, an error margin requires that $o_{\sim T\max} + \mu < o_{T\max}$. Conversely, for a competing node k with output o_k, the inequality $o_k + \mu < o_{T\max}$ must be satisfied for no error signal to be backpropagated from k.

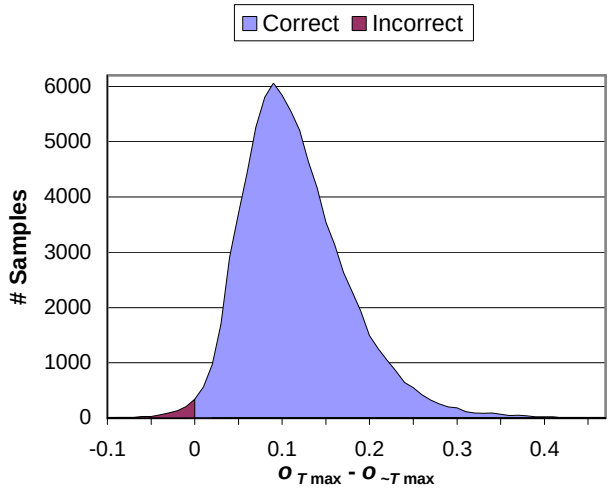

Figure 1. Network output margin of error after lazy training.

Requiring an error margin is important since the goal of learning in this instance is not simply to learn the training environment well but to be able to generalize. This is especially important in the case of noisy problem data. During the training process, μ can be increased gradually and might even be negative to begin with, not expressly requiring

correct classification at first. This gives the network time to configure its parameters in a more uninhibited fashion. Then μ is increased to an interval sufficient to account for the variance that appears in the test data, allowing for robust generalization.

At the extreme value of μ equal to 1, lazy training becomes standard SSE training, with output values of 1.0 and 0.0 required to satisfy the margin. Since a margin of 1 can never be obtained without infinite weights, an error signal is always backpropagated on every pattern.

3 Softprop Heuristic

The softprop heuristic performs a novel explore-exploit search of the solution space for multi-layer neural networks. Softprop exchanges the use of a single pure objective function with a mixture taking advantage of both lazy training and SSE minimization at appropriate times during the learning process. The heuristic is as follows:

> For each epoch, let the lazy training error margin $\mu = t/T$, where $t \in \{0, 1, 2, \ldots\}$ is the current epoch and T is the maximum number of epochs to train.

Softprop causes a smooth shift from lazy training to SSE minimization as the search progresses. The lazy exploration phase first steers the decision surface toward a general problem solution without saturating network weights prematurely. Then, as learning tends toward SSE exploitation, the distance of the decision boundary from proximate

patterns is maximized. The practical aspect of this approach is analogous to simulated annealing, where a Boltzmann stochastic update is used with an update probability "temperature" that is gradually reduced to allow the network to gradually settle into an error minimum.

The complexity of softprop is equivalent to that of standard SSE optimization and lazy training and converges in comparatively as many epochs.

4 Experiments

Empirical results are presented in this section.

4.1 Data sets

Several well-known benchmark classification problems were selected from the UC Irvine Machine Learning Repository (UCI MLR). The problems were selected so as to have a wide variety of characteristics (size, number of features, complexity, etc.) in order to demonstrate the robustness of the learning algorithms. Results on each problem were averaged using 10-fold stratified cross-validation.

4.2. Training parameters

Experiments were performed comparing the SSE and lazy training objective functions against the proposed softprop heuristic. Feed-forward multi-layer perceptron networks with a single, fully-connected hidden layer were trained through on-line backpropagation.

In all experiments, weights were initialized to uniform random values within the range [-0.3,0.3]. The learning rate was 0.1 and momentum was 0.5. Networks trained to optimize SSE used an error threshold (d_{max}) of 0.1.

Feature values (both nominal and continuous) were normalized between zero and one. Training patterns were presented to the network in a random order each epoch. The same initial random seed for network weight initialization and sample shuffling was used for all experiments on a given data set.

SSE and lazy training continued until the training set was successfully learned or until training classification error ceased to decrease for a substantial number of epochs. The softprop schedule was set for an equivalent number of epochs. A holdout set (between 10-20% of the data) was randomly selected from the training set each fold to perform model validation. The model selected for test evaluation was the network epoch with the best holdout accuracy.

Network architecture was optimized to maximize generalization for each problem and learning heuristic. Pattern classification was determined by *winner-take-all* (the class of the highest outputting node is chosen) on all models tested.

5 Results

Table 1 lists the results of a naïve Bayes classifier (taken from [21]), standard SSE backpropagation, lazy training, and softprop on the selected UCI MLR corpus. Each field lists first the average holdout set accuracy using 10-fold stratified cross validation. The second value is the variance of the classification accuracy over all ten runs. The best generalization and variance for each problem is bolded.

On average, an optimized backpropagation network minimizing SSE is superior to a naïve Bayes learner on the above classification problems. Lazy training obtains a significantly higher accuracy over SSE training. Interestingly, the SSE minimizing network achieves an SSE up to two orders of magnitude lower than that of the selected lazy trained network, a moot point because SSE is simply a means to an end, not the ultimate measure of optimality. However, this serves to illustrate that the SSE and lazy approaches each perform radically different searches of the problem space.

Softprop performed better than both lazy training and simple SSE backpropagation, reducing classification error by 17.1% and had the best overall accuracy. Softprop is particularly effective in learning noisy problems (e.g. *sonar*) where premature saturation of weights could trap the network in a local minimum.

Decreasing classification error is a worthy achievement, but of possibly even greater import is the fact that softprop has a significant overall reduction in the variance of

classification error over the ten cross-validation folds. Lazy training shows a minor overall reduction in standard deviation of error over SSE backpropagation. Softprop provides a larger reduction of 38.6%. This supports the softprop approach as being more robust.

Table 1. Results on UCI MLR data sets using 10-fold stratified cross-validation.

Data set	**Bayes**	**SSE**	**Lazy**	**Softprop**
ann	**99.7** **0.1**	98.25 0.54	97.92 0.55	98.29 0.43
bcw	93.6 3.8	96.78 2.05	96.87 3.76	**97.07** **1.61**
ionosphere	85.5 4.9	88.03 6.12	**90.60** **4.80**	89.17 4.93
iris	94.7 6.9	93.33 7.30	**95.33** 4.27	**95.33** **3.06**
musk2	97.1 0.7	99.38 **0.21**	**99.44** 0.40	99.23 0.48
pima	72.2 6.9	**77.47** 3.75	76.69 5.22	76.69 **2.37**
sonar	73.1 11.3	77.40 10.77	81.73 14.08	**83.65** **8.67**
wine	94.4 5.9	94.94 8.04	96.63 4.58	**98.88** **2.29**
Average	88.79 5.06	90.70 4.85	91.93 4.74	**92.29** **2.98**

6 Conclusions and Future Work

The softprop heuristic of gradually increasing the required margin of error between classifier outputs, reflecting a steady shift between classification error exploration and SSE exploitation, was shown to be superior to either optimization of SSE or classification

error alone. Softprop reduces classification error over a corpus of machine learning data sets by 17.1% and variance in test accuracy by 38.6%.

While the parameters of the SSE backpropagation learner had been extensively optimized, due to time constraints little parameter tuning was done on the softprop heuristics. It is possible that by optimizing the learning parameters even more significant improvements could be shown. Providing specialized exploration policies for local areas of the parameter space by dynamically setting a particular μ for each pattern will be considered. In this way, local learning can proceed at different speeds depending on the local characteristics of the problem domain. As learning progresses, the values for the local μ can be learned and refined according to need. We will experiment with the feasibility of relaxing the restrictions of our search by allowing a negative-valued μ. This in essence provides a way to "tunnel" through difficult, inconsistent, or noisy portions of the problem space in order to escape local minima and might assist in achieving more optimal solutions.

References

[1] Aha, David W., editor, *Lazy Learning*, Kluwer Academic Publishers, Dordrecht, May 1997.

[2] Andersen, Tim and Tony R. Martinez, "Cross Validation and MLP Architecture Selection", *Proceedings of the IEEE International Joint Conference on Neural Networks IJCNN'99*, CD Paper #192, 1999.

[3] Andersen, Tim and Martinez, Tony, "Wagging: A learning approach which allows single layer perceptrons to outperform more complex learning algorithms", *Proceedings of the IEEE International Joint Conference on Neural Networks IJCNN'99*, CD Paper #191, 1999.

[4] Bartlett, Peter L., "The Sample Complexity of Pattern Classification with Neural Networks: The Size of the Weights is More Important than the Size of the Network", *IEEE Trans. Inf. Theory*, 44(2), 1998, pp. 525-536.

[5] Barnard, Etienne, "Performance and Generalization of the Classification Figure of Merit Criterion Function", *IEEE Transactions on Neural Networks*, 2(2), March 1991, pp. 322-325.

[6] Castellano, G., A. M. Fanelli and M. Pelillo, "An empirical comparison of node pruning methods for layered feed-forward neural networks", *Proc. IJCNN'93-1993 Int. J. Conf. on Neural Networks*, Nagoya, Japan, 1993, pp. 321-326.

[7] Castellano, G., A. M. Fanelli, and M. Pelillo, "An iterative pruning algorithm for feed-forward neural networks", *IEEE Transactions on Neural Networks*, vol. 8 (3), 1997, pp. 519-531.

[8] Hampshire II, John B., "A Novel Objective Function for Improved Phoneme Recognition Using Time-Delay Neural Networks", *IEEE Transactions on Neural Networks*, Vol. 1, No. 2, June 1990.

[9] Simon, H., "Theories of decision-making in economics and behavioral science," *American Economic Review*, XLIX (1959), 253.

[10] Maclin, R and Opitz, D, "An empirical evaluation of bagging and boosting", *The Fourteenth National Conference on Artificial Intelligence*, 1997.

[11] Mitchell, Tom, *Machine Learning.* McGraw-Hill Companies, Inc., Boston, 1997.

[12] Rimer, M., Andersen, T. and Martinez, T.R., "Improving Backpropagation Ensembles through Lazy Training," Proceedings of the *IEEE International Joint Conference on Neural Networks IJCNN'01*, pp. 2007-2112, 2001.

[13] Rimer, Michael, "Lazy Training: Interactive Classification Learning," Masters Thesis, Brigham Young University, April 2002.

[14] Rimer, M. Martinez, T.R. and D. R. Wilson, "Improving Speech Recognition Learning through Lazy Training," to appear in Proceedings of the *IEEE International Joint Conference on Neural Networks IJCNN'02*.

[15] Rumelhart, David E., Hinton, Geoffrey E. and Williams, Ronald J., "Learning Internal Representations by Error Propagation", Institute for Cognitive Science, University of California, San Diego; La Jolla, CA, 1985.

[16] Schiffmann, W., Joost, M. and Werner, R., "Comparison of Optimized Backpropagation Algorithms", *Artificial Neural Networks*, European Symposium, Brussels, 1993.

[17] Schiffmann, W., Joost, M. and Werner, R., "Optimization of the Backpropagation Algorithm for Training Multilayer Perceptions", University of Koblenz: Institute of Physics, 1994.

[18] Wang, C., Venkatesh, S. S., and Judd, J. S., "Optimal stopping and effective machine complexity in learning", in Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems*, vol. 6, Morgan Kaufmann, San Francisco, 1994, pp. 303-310.

[19] Watkins, C., and Dayan, P. "Q-learning", *Machine Learning*, vol. 8, 1992, pp. 279-292.

[20] Werbos, P., "Backpropagation: Past and future", *Proceedings of the IEEE International Conference on Neural Networks*, IEEE Press, 1988, pp. 343-353.

[21] Zarndt, Frederick, "A Comprehensive Case Study: An Examination of Machine Learning and Connectionist Algorithms," Masters Thesis, Brigham Young University, 1995.

Chapter 5

CB3: An Adaptive Error Function for Backpropagation Training

Michael Rimer and Tony Martinez
Computer Science Department
Brigham Young University
3361 TMCB, PO Box 26576
Provo, UT 84602-6576, USA
e-mails: mrimer@axon.cs.byu.edu, martinez@cs.byu.edu

Abstract. Effective backpropagation training of multi-layer perceptrons depends on the incorporation of an appropriate error or objective function. Classification-based (CB) error functions are heuristic approaches that attempt to guide the network directly to correct pattern classification rather than using common error minimization heuristics, such as sum-squared error and cross-entropy, which do not explicitly minimize classification error. This work presents CB3, a novel CB approach that learns the error function to be used while training. This is accomplished by learning pattern confidence margins during training, which are used to dynamically set output target values for each training pattern. On eleven applications, CB3 significantly outperforms previous CB error functions, and also reduces average test error over conventional error metrics using 0-1 targets without weight decay by 1.8%, and by 1.3% over metrics with weight decay. CB3 also exhibits lower model variance and tighter mean confidence interval.

Key words. neural network, backpropagation, classification, error functions, adaptive targets

Abbreviations. CB – classification-based; CE – cross-entropy; SSE – sum-squared error

1 Introduction

Multi-layer feed-forward neural networks trained through error backpropagation [11] have received substantial attention as robust learning models for classification tasks. Classification-based (CB) error functions [9, 10] are a relatively new method of training multi-layer perceptrons. CB functions heuristically seek to directly minimize classification error by backpropagating network error only on misclassified patterns. In doing so, they perform relatively minimal updates to network parameters in order to discourage premature weight saturation and overfitting. This is conducive to higher accuracy in classification problems than optimizing with respect to commonly used error functions, such as sum-squared error (SSE) and cross-entropy (CE). This work presents a novel CB error function, CB3, which improves on existing CB functions. It is an adapting error function that dynamically sets output target values during training by learning confidence margins on each pattern in the training set. These confidence margins guide the network in learning each pattern according to the ability of the network, selectively learning patterns that appear to provide better generalization while avoiding those that would encourage weight saturation and possible overfit without improving generalization.

Performance of networks trained with the CB3 error function is compared against previous CB error functions, SSE and CE with and without weight decay [7], on a corpus of eleven benchmark machine learning datasets. CB3 shows a significant reduction in average test error of 1.8% over standard backpropagation using conventional 0-1 target

values without weight decay, and a significant decrease of 1.3% average test error over backpropagation augmented by weight decay regularization. CB3 also exhibits lower model variance with smaller standard deviation.

Section 2 reviews related work and motivation for this new approach. Sections 3 and 4 present the CB3 algorithm and a working example. Section 5 describes experiments performed and Section 6 gives empirical results and discussion.

2 Motivation for CB3

A prime goal of classification learning is to achieve high recognition rates on unseen data. To generalize well, a learner must use a proper objective function. The validity of using common differentiable metrics like sum-squared-error (SSE) relies on the assumption that sample outputs are offset by inherent Gaussian noise, being normally distributed about a cluster mean. For function approximation of an arbitrary signal, this presumption often holds. However, this assumption is invalid for classification tasks, where assigned real-valued target vectors are arbitrary values used to represent class labels. This suggests that other error metrics (e.g. cross-entropy) are more suited to classification problems. Likewise, cross-entropy (CE) is preferable to SSE when output class distributions are not balanced. When this is not the case, CE and SSE may perform equivalently.

Traditionally, classification problems are learned through error backpropagation by providing a vector of strict ("hard") 0/1 target values to represent the class label of a particular pattern. Minimizing an error function with hard target values tends to a saturation of weights, often equated with overfitting. There is evidence that the magnitude of the weights in a network plays a more important role in generalization than the number of hidden nodes [1]. It follows that overfit might be reduced by keeping the weights smaller. Regularization methods such as weight decay [7,14] are commonly used to discourage weight saturation and overfit. These methods generally assume that overfitting is a global phenomenon. However, overfit can vary significantly in different regions of the model. Proper early stopping methods that take advantage of this information can further improve generalization [6].

An alternate method of discouraging weight saturation is to provide a maximum error tolerance threshold, d_{max}, and not backpropagate any error when network output values are within this range of the target values. That is, for a given d_{max}, target value, t_k, and network output, o_k, no weight update occurs if the absolute error $| t_k - o_k | < d_{max}$. This threshold is arbitrarily chosen to indicate the point at which a sample has been sufficiently approximated. Using an error threshold, a network is permitted to converge with smaller weights [12].

Rankprop [5] provides an alternative method to training with hard target values and empirically shows that it improves generalization. Rankprop records the output of the learner for each training pattern. It then sorts the samples in the training set based on

class, then according to output values. Thus, a rank of the samples consistent with the current model is developed and used to define the target values on the next epoch. The idea behind Rankprop is that in the case of complex nonlinear solutions a simpler, *less nonlinear* function is provided to learn instead. The resulting simpler model often generalizes better.

Prior work has shown [8, 9, 10] that methods of calculating softer values for each training pattern based on the network's output vector improves generalization and reduces variance on classification problems over a corpus of benchmark learning problems. One of these, called lazy training or CB1, focuses on classification accuracy backpropagates an error signal through the network only when a pattern is misclassified. CB2, starts with the "lazy" targets used in CB1 and gradually separates them until they reach the 0-1 targets used in standard training. Other approaches involve using an "oracle" teacher network to provide target output values to simpler networks that can learn to emulate its behavior.

This work extends CB1 and CB2 by providing a heuristic to learn how much error can be tolerated in each training pattern based on how well the network is learning in order to improve generalization.

3 CB3 Algorithm

Learning how much error to backpropagate based on the performance of the network being trained is, in effect, a meta-learning algorithm. In other words, the error function

itself is learned based on the ability of the network to learn it. CB3 accomplishes this by learning how confident the network is in classifying each training pattern as learning progresses. This method for dynamically learning pattern confidence margins is shown in Figure 1. CB3 modifies the standard back-propagation algorithm in the following three ways:

- For each pattern-output node pair, a confidence value is stored and modified over time. This value represents an interval in the range of the squashing function that reflects the numeric amount by which the node is assisting in classifying the pattern correctly or incorrectly.
- As training progresses, each pattern's learned confidence values are used in calculating the target output values for the pattern.
- The objective function is modified based on these target values to decide how large of an error signal to backpropagate through the network for each pattern.

Without loss of generality, in this work it is assumed that a single, distinct output node in the network represents each class label. Let N be the number of output nodes (and distinct class labels). On a given pattern, let o_j be the output value of the j^{th} output node of the network ($0 < o < 1$, $1 \leq j \leq N$). Let T designate the target output class for that pattern and c_j signify the class label of the j^{th} output node. For the output node corresponding to the pattern's class label, $c_j = T$. We refer to this output node as c_T for short. For non-target output nodes, $c_j \neq T$. Non-target output nodes are called

competitors. Let o_T denote the value of the target output node. Let $o_{\sim T\max}$ denote the value of the highest-outputting competitor.

Initialization.
Set expected confidence values, $C_{i,j}$, for each pattern-output node *ij* pair.

Training.
Present a training pattern, *i*, to the network.
Determine o_T and $o_{\sim T\max}$.

1. For each output node *j*, set its target output, T_j:

$$T_j \equiv \begin{cases} \min(1, o_{\sim T\max} + \alpha C_{i,j}) & \text{if } c_j = T \text{ and } C_{i,j} > 0 \\ \max(0, o_T - \alpha C_{i,j}) & \text{if } c_j \neq T \text{ and } C_{i,j} > 0 \\ o_{\sim T\max} & \text{if } c_j = T \text{ and } C_{i,j} \leq 0 \\ o_T & \text{if } c_j \neq T \text{ and } C_{i,j} \leq 0 \end{cases} \quad (1)$$

2. From each output node *j*, backpropagate error, ε_j.

$$\varepsilon_j \equiv \begin{cases} T_j - o_j & \text{if } c_j = T \text{ and } o_j < T_j \\ T_j - o_j & \text{if } c_j \neq T \text{ and } o_j > T_j \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

3. For all *j*, update confidence value $C_{i,j}$ to make it closer to the observed interval.

$$C_{i,j} = C_{i,j} + \eta_C (OI_j - C_{i,j}) \quad (3)$$

where the observed interval, OI_j, is defined as

$$OI_j \equiv \begin{cases} o_j - o_{\sim T\max} & \text{if } c_j = T \\ o_T - o_j & \text{if } c_j \neq T \end{cases} \quad (4)$$

Continue training until stopping criterion is satisfied.

Figure 1. CB3 algorithm.

In the initialization phase, for each combination of training pattern and network output node, a confidence value, *C*, is stored. With *I* training patterns and *J* output nodes in the network, this results in *IJ* values being stored. These values indicate the amount of

confidence the network's output nodes expect to have in classifying the corresponding pattern correctly. Positive values indicate that a pattern is expected to be correctly classified by the output node while negative values mean it is expected to be misclassified by the node. These values are updated as training progresses. We have found initial confidence values above approximately 0.2 to generalize better than values below 0.2 on all applications we tested (see Figure 2).

During training, patterns are shuffled each epoch and stochastically presented to the network. The vector of target values for a pattern's class outputs is determined as shown in Figure 1, equation 1. Each target value is calculated differently depending on whether $c_j = T$ and whether the confidence value for that node, $C_{i,j}$, is positive or negative. With a positive confidence, the target value for c_T is set to $o_{\sim T\max} + \alpha C_{i,j}$. That is, it is set to the maximum competitor's output value plus the confidence value on node j for pattern i, multiplied by α, a multiplicative factor greater than or equal to one. This factor intuitively refers to how aggressively CB3 will try to separate the target values for opposing classes. A value of one will allow targets to remain closer together while a greater value will separate them more. Conversely, competitor class targets are set to $o_T - \alpha C_{i,j}$, i.e., c_T's output value minus the confidence value, multiplied by α. We have found values for α above 1.5 to produce better generalization than lower α values on all tested applications (see Figure 3).

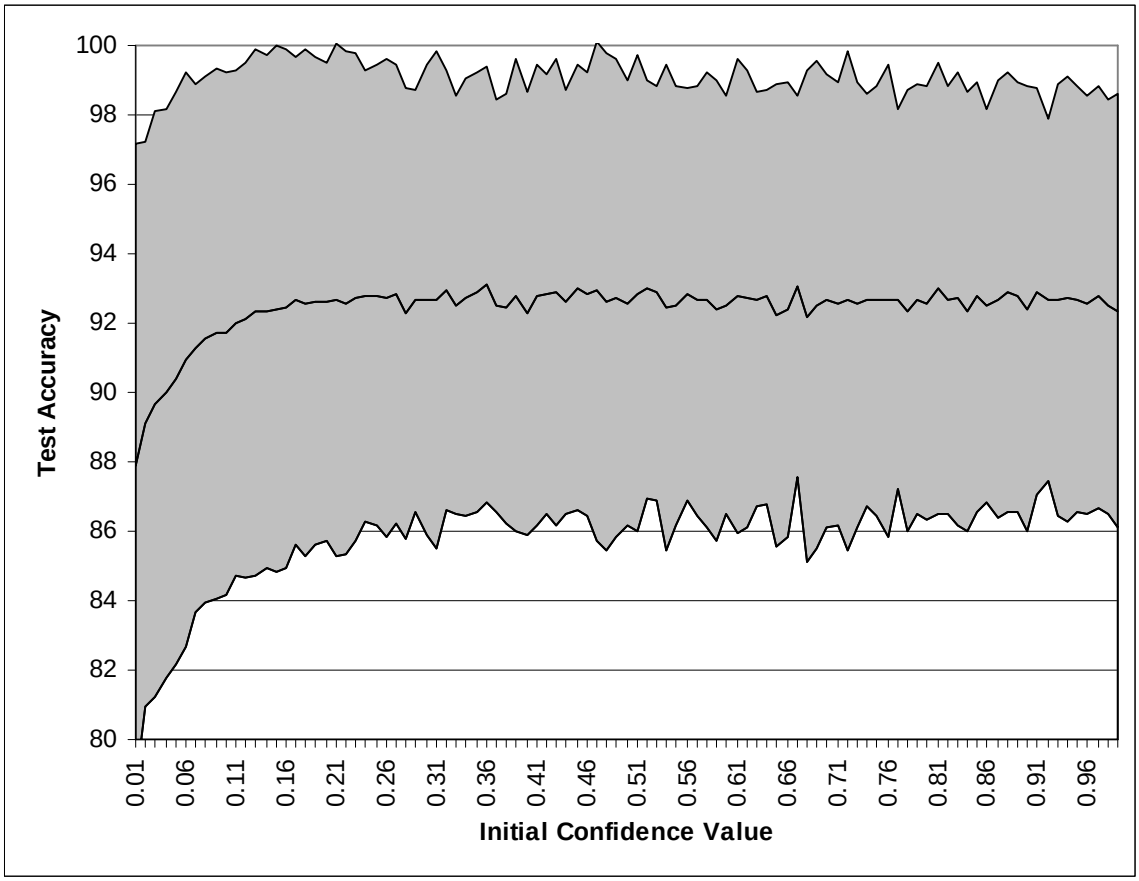

Figure 2. Influence of CB3's initial confidence values on the test accuracy of selected applications. The highlighted area is the 95% confidence interval for an observation.

The min(1,·) and max(0,·) operators are used to keep the target values within the range of possible output values (*e.g.* [0,1] for the sigmoid activation function).

When a negative confidence value exists, $o_{\sim T\max}$ and o_T are used as the target values. The reason for this is that a negative confidence indicates that this output node has learned to consistently misclassify this pattern over time. This could happen if the network either does not have enough hidden nodes to learn the problem sufficiently, or a noisy or incorrectly labeled pattern is encountered. In either case, further effort to learn these

"problem" patterns could lead to premature weight saturation or overfit. On the other hand, it is possible that the network's hidden nodes have simply not yet learned to model this area of the problem space correctly, in which case some effort should still be made to learn to classify this pattern correctly. If this is not the case, however, undue resources (in the form of network parameters) have not been squandered in trying to learn a pattern that will probably not improve generalization and could even be detrimental to it.

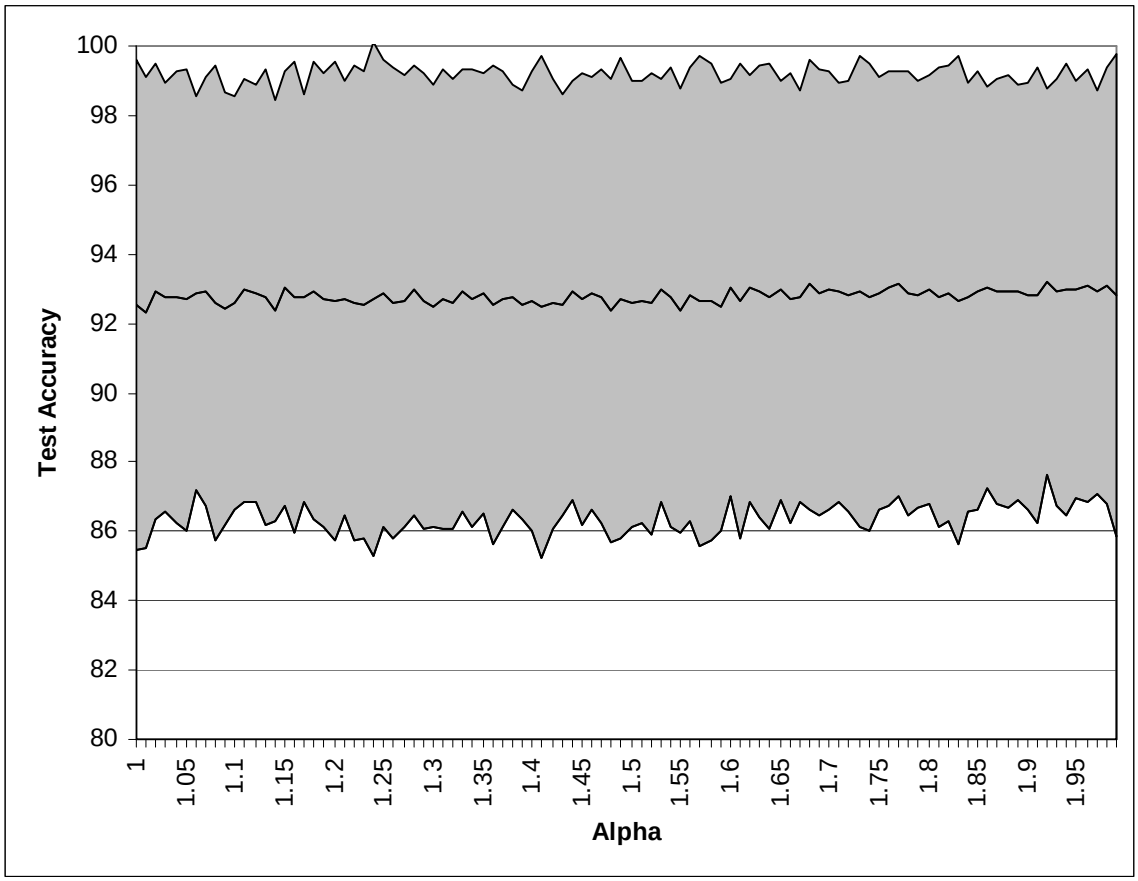

Figure 3. Influence of CB3's α parameter on the test accuracy of selected applications. The highlighted area is the 95% confidence interval for an observation.

Once the target vector has been calculated, the error signal for backpropagation through the network is determined (see Figure 1, equation 2). Observe that no error is backpropagated by CB3 when an output value is already better than the dynamically set target value (i.e., a value higher than the target value for the target class node, or a value lower then the target for nodes of other classes). That is, backpropagating error depends on whether the pattern is currently being classified correctly or not, as in CB1 [10]. This is akin to setting a d_{max} error threshold (see section 2) for each pattern based on the value of the highest outputting competitor. This selective error signal avoids updating network weights when doing so would not necessarily lead to improved accuracy (the pattern is already being classified correctly) while risking premature weight saturation and overfit. On misclassified patterns, error is backpropagated only from output nodes that fall short of their dynamic target values and are considered in some way responsible for the misclassification.

Following the determination and backpropagation of error terms comes the meta-learning step, where the learned confidences used in calculating the dynamic target values are also iteratively updated (see Figure 1, equation 3). The current pattern's observed interval vector, OI_i, is calculated as shown in Figure 1, equation 4, indicating the amount of split between the output values of positive and negative class labels on this pattern, respective to each output node.

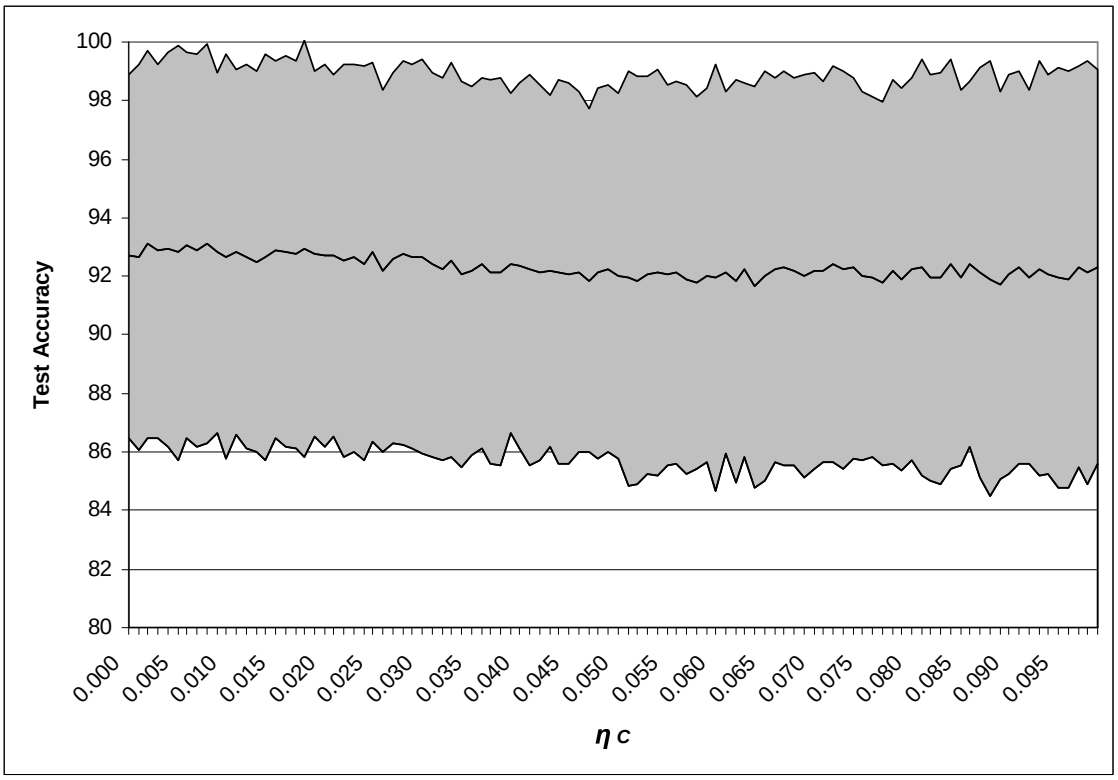

Figure 4. Influence of η_C on the test accuracy of selected applications. The highlighted area is the 95% confidence interval for an observation.

As an error delta is calculated for iterative weight updating to bring output values closer to the target values, likewise the learned confidence value $C_{i,j}$ is subtracted from the actual observed interval OI_j and multiplied by learning rate η_C to calculate a confidence value delta. This confidence delta is added to the learned confidence. We observed values of η_C less than 0.025 to have the highest generalization and lowest variance on all tested applications (see Figure 4).

As training continues, $C_{i,j}$ will be iteratively learned. It reflects the ability of the network output nodes to locally distinguish the target class from the others. Learning this value

has the practical purpose of hinting to the network which patterns can be classified with confidence (such as cluster centers), which patterns need to be learned more (fringe patterns), and which patterns appear difficult to learn correctly with the current network at all (noise or overlapping classes). This enables the training process to guide the network to spend more resources on learning training patterns that most probably contribute to higher generalization accuracy while selectively ignoring those that lead the network to overfit and weight saturation as it attempts to learn them.

CB3 requires an O(n) scan through the n network outputs to determine the highest target and competitor values, to set the target value for each node before backpropagation, and then to update the observed confidence interval following backpropagation. However, this additional overhead to the standard backpropagation is negligible compared to the computation requirements of O(ih) for feed-forwarding a pattern vector and O(ihn) for backpropagation, where i is the number of inputs and h is the number of hidden nodes. In fact, CB3 saves time by omitting the error backpropagation step for correctly classified patterns with sufficient output confidences. The number of epochs required to converge is similar for CB3 and CE training, and CB3 generally converges in about half as many epochs as SSE training.

The following section presents a hypothetical example of using the CB3 algorithm.

4 CB3: An Example

Given a three-class problem with class labels A, B, and C, a pattern *i* is labeled as type B. Let $\alpha = 1.5$ and $\eta_C = 0.01$.

Let the current confidence values for this pattern be [0.1, 0.3, 0.1], indicating the network output nodes all believe they are performing correctly but with only a moderate margin separating them. If the network were to output [0.4, 0.7, 0.3] on this pattern, o_T is 0.7 and $o_{\sim T\max}$ is 0.4. The target vector *T* would be calculated to be

$$T = [o_T - mC_{i,j} , o_{\sim T\max} + \alpha C_{i,j} , o_T - \alpha C_{i,j}]$$
$$= [0.7 - 1.5(0.1) , 0.4 + 1.5(0.3) , 0.7 - 1.5(0.1)]$$
$$= [0.55, 0.85, 0.55].$$

All values are within the imposed ranges, so the min/max operators are omitted here for clarity. Next, the error vector ε is calculated. The first and third outputs are satisfactorily outside the learned confidence margins but the second node has a target value exceeding its output. Thus ε is [0,0.15,0]. Error is backpropagated from the second output node only. Its observed confidence may increase over subsequent iterations.

The pattern's confidence values are now each updated by the delta η_C $(OI_j - C_{i,j})$, where the observed interval vector *OI* is

$$OI = [o_T - o_j \, , \, o_j - o_{\sim T\max} \, , \, o_T - o_j]$$

$$= [0.7 - 0.4, 0.7 - 0.4, 0.7 - 0.3]$$

$$= [0.3, 0.3, 0.4]$$

and the update delta is

$$= [0.01(0.3 - 0.1), 0.01(0.3 - 0.2), 0.01(0.4 - 0.1)]$$

$$= [0.002, 0.001, 0.003].$$

The learned confidences are increased to [0.102, 0.201, 0.103]. As training progresses, if the network continues to output similar values, the confidences on this pattern will continue to grow. As these confidences get large, greater error is backpropagated and the observed interval over network outputs will tend to reflect these confidences.

5 Experiments

Several well-known benchmark classification problems were selected from the UC Irvine Machine Learning Repository (UCI MLR) [2]. The problems were selected so as to have a wide variety of characteristics (number of patterns, number and type of features, and complexity) in order to analyze the robustness of the learning algorithm.

Experiments were performed using a privately developed C++ library comparing feed-forward multi-layer perceptron networks optimizing SSE and cross-entropy (CE), both with standard 0-1 targets with and without weight decay regularization, to the CB1-3 algorithms. It was observed that SSE and CE optimizing networks yielded nearly identical results using both static 0-1 targets and with the CB1-3 algorithms. Therefore, only the results of training with CE are presented for brevity.

The multi-layer perceptrons had a single, fully connected hidden layer and were trained through on-line backpropagation. The optimal number of hidden nodes was empirically determined for each task based on holdout set accuracy, searching layer sizes within the range from one to fifty hidden nodes. In all experiments, network weights were initialized to uniform random values within the range [-0.1,0.1] [13]. The learning rate was 0.1 and momentum was 0.7. Weight decay values between $\lambda = 0.00001$ to 0.0001 were used, optimized for each application [7]. For CB3, $\alpha = 1.5$, $\eta_C = 0.01$, and the initial pattern confidence values were 0.25.

Feature values (both nominal and continuous) were normalized between zero and one. Training patterns were presented to the network in a random order each epoch. The same initial random seed for network weight initialization and sample shuffling was used for all experiments on a given dataset.

Pattern classification was determined by *winner-take-all* (the class of the highest outputting node is chosen). Training continued until the training set was successfully

learned or training set classification error ceased to decrease for a substantial number of epochs. The resultant number of epochs trained was comparable among all approaches within a factor of three. The model selected for test evaluation was the network on the epoch with the best holdout set accuracy, where the holdout set consisted of 20% of the original training data.

6 Results and Discussion

Table 1 lists the results of testing a multi-layer perceptron backpropagating error maximizing cross-entropy without weight decay (BP), with weight decay (BPw), and CB1-3 on the selected applications. Each field lists first the average test set accuracy using 30-fold stratified cross validation. Neural network experiments were averaged over 30 runs with random initial weights. The first value in each cell is the average accuracy over these runs. The second value is the 95% Student's *t* confidence interval for these means. The best generalization for each problem is underlined.

Table 1. Results on UCI MLR datasets using stratified cross-validation.

Data set	ann	bal-ance	bcw	derm	ecoli	ionos	iris	musk2	pima	sonar	wine	**avg**
BP	98.1 0.18	95.0 1.80	97.1 1.25	97.2 1.66	85.6 3.89	91.6 2.90	94.9 3.72	<u>99.4</u> 0.13	72.1 3.19	80.6 3.21	98.5 1.80	91.8 0.37
BPw	95.2 0.21	96.6 1.59	96.7 0.74	97.2 1.72	86.1 3.82	92.8 1.65	96.7 1.54	97.0 0.25	75.6 1.63	83.2 2.88	98.1 1.09	92.3 0.36
CB1	97.4 0.30	<u>97.4</u> 0.89	97.2 1.10	96.1 2.00	84.2 4.30	90.6 2.82	96.7 2.97	99.2 0.15	76.3 2.63	84.1 3.00	97.8 2.99	92.5 0.37
CB2	98.2 0.19	97.1 1.25	96.9 1.14	<u>97.8</u> 1.34	86.0 4.30	92.0 2.31	96.0 3.48	99.3 0.10	75.5 2.88	83.7 2.27	98.3 2.88	92.8 0.37
CB3	<u>98.3</u> 0.15	97.2 1.00	<u>97.5</u> 0.81	<u>97.8</u> 1.39	<u>86.6</u> 3.70	<u>92.9</u> 2.43	<u>97.3</u> 2.51	98.9 0.21	<u>78.1</u> 2.78	<u>86.1</u> 2.32	<u>98.6</u> 1.71	<u>93.6</u> <u>0.35</u>

CB3 has higher test accuracy and tighter mean confidence interval then CE without weight decay (BP) on ten of the eleven datasets tested. BP outperformed CB3 on the *musk2* dataset. CB3 reduces average test error by 1.8% over BP, significant with a pairwise Student's *t* confidence of $p < 0.05$. CB3 has a tighter confidence interval, which indicates it has a smaller standard deviation and is more robust to perturbations in the initial network parameter values and pattern presentation order.

CB3 has higher accuracy than BPw on all eleven datasets and exhibits an average decrease in test error of 1.3%, significant with a pairwise *t* confidence of $p < 0.05$. CB3 has a tighter confidence interval than BPw on six of the eleven datasets and is slightly higher than BPw on average. CB3 outperforms CB1 and CB2 on nine of eleven datasets, with an average decrease in test error of 1.1% and 0.8%, respectively, significant with a pairwise *t* confidence of $p < 0.05$. CB3 has a tighter confidence interval than CB1 and CB2.

For a given function *f(x)*, there may exist a function *g(x)* that also solves the given problem but is easier for backpropagation to learn [3, 4, 5]. Recall that CB3 does not modify the actual backpropagation algorithm used for updating the network parameters in any way. CB3's power comes through the modification of target values as a tool for smoother training. Most often, conventional target values of 0 and 1 are used in classification tasks to learn proper class labels. CB3, using target values greater than 0

and less than 1, consequently calculates much smaller error terms during the initial phase of training. Less error may result in a function that is easier for backpropagation to learn.

7 Conclusions and Future Work

CB3 is shown to be superior to multi-layer backpropagation networks trained with previous CB error functions and CE with hard targets without weight decay (BP) and with weight decay (BPw) over a corpus of eleven applications. CB3 significantly reduces average test error by 1.8% over BP, by 1.3% over BPw, and by 1.1% and 0.8% over CB1 and CB2, respectively. It is surmised that CB3, learning to approximate iteratively learned target values, provides a function that is easier for backpropagation to learn than the strict conventional 0-1 classification function.

Since the learned confidence values are able to implicitly represent noisy patterns, they could be used to explicitly mark overlapping class regions. This knowledge is useful to reduce local uncertainty for problems with regions of the feature space that are inherently multi-class. A method for such an approach and analysis of its efficacy is forthcoming. An ROC analysis of CB3 is planned on applications where the cost of false positives is different than false negatives.

Furthermore, preliminary tests have shown that pattern misclassifications are not highly correlated between CB3 and BP. We will experiment with combining BP- and CB-

trained networks in hybrid ensembles and voting committees to further improve generalization.

References

1. Bartlett, P. L. (1998). The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network, *IEEE Transaction Information Theory* 44(2), pp. 525-536.
2. Blake, C. and Merz, C. (1998). UCI machine learning repository, http://www.ics.uci.edu/~mlearn/MLRepository.html. Irvine, CA: University of California, Department of Information and Computer Science.
3. Caruana, R. (1996). Algorithms and applications for multitask learning, In: *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 87-95, Bari, Italy.
4. Caruana, R. (1997). Multitask Learning. Ph.D. Dissertation, School of Computer Science, Carnegie Mellon University. Pittsburg, PA.
5. Caruana, R., Baluja, S., & Mitchell, T. (1996). Using the future to 'sort out' the present: rankprop and multitask learning for medical risk evaluation, In: *Advances in Neural Information Processing Systems 8*, pp. 959-965, MA: MIT Press, Cambridge.
6. Caruana, R., Lawrence, S. & Giles, C. (2000). Overfitting in neural networks: backpropagation, conjugate gradient, and early stopping, *Neural Information Processing Systems*, Denver, Colorado.

7. Krogh, A. & Hertz, J. (1992). A simple weight decay can improve generalization, *Advances in Neural Information Processing Systems 4*, pp. 950-957, San Mateo, CA.
8. Menke, J., Peterson, A., Rimer, M. & Martinez, T. (2002). Neural network simplification through oracle learning, In: *Proceedings of the IEEE International Joint Conference on Neural Networks IJCNN'02*, pp. 2482-2497.
9. Rimer, M. and Martinez, T. (2004). Softprop: softmax neural network backpropagation learning, In: *Proceedings of the IEEE International Joint Conference on Neural Networks IJCNN'04*, pp. 979-984.
10. Rimer, M. and Martinez T. (2006). Classification-based objective functions, *Machine Learning* 63(2), pp. 183-205.
11. Rumelhart, D. E., Hinton, G. and Williams, R. (1985). *Learning Internal Representations by Error Propagation*, Institute for Cognitive Science, University of California, San Diego; La Jolla, CA.
12. Schiffmann, W., Joost, M. and Werner, R. (1994). *Optimization of the Backpropagation Algorithm for Training Multilayer Perceptions*, University of Koblenz: Institute of Physics.
13. Thimm, G. and Fiesler, E. (1997). High-order and multilayer perceptron initialization, *IEEE Transactions on Neural Networks* 8(2), 249-259.
14. Werbos, P. (1988). Backpropagation: past and future, In: *Proceedings of the IEEE International Conference on Neural Networks*, IEEE Press, New York, pp. 343-353.

Chapter 6

Analysis of Classification-based Error Functions

MICHAEL RIMER mrimer@axon.cs.byu.edu
TONY MARTINEZ martinez@cs.byu.edu
Computer Science Department, Brigham Young University, Provo, UT 84602, USA
Phone: (801) 422-6464 Fax: (801) 422-0169

Abstract. Effective backpropagation training of multi-layer perceptrons depends on the incorporation of an appropriate error or cost function. Classification-based (CB) error functions are heuristic approaches that attempt to guide the network directly to correct pattern classification rather than using common error minimization heuristics, such as sum-squared error (SSE) and cross-entropy (CE), which do not explicitly minimize classification error. This work presents a comparative study of SSE, CE, CB1, CB2, and CB3 error functions on a corpus of machine learning applications. It is demonstrated that CB3 achieves significantly higher generalization and lower variance than the other error functions on these datasets. Further analysis shows CB3 to be more robust to initial network parameter settings, pattern presentation order, learning rate, the number of hidden nodes, and the ability to avoid weight saturation during training. This suggests CB3 is capable of performing well while requiring a minimum of learning parameter tuning.

Keywords: neural networks; backpropagation; classification; error functions; adaptive targets

1 Introduction

Artificial neural networks (ANNs) have received substantial attention as robust learning models for classification learning tasks and have been shown to perform comparably to other learning algorithms (Caruana & Niculescu-Mizil, 2006). Classification-based (CB) error functions (Rimer & Martinez, 2004, Rimer & Martinez, 2006a; Rimer & Martinez, 2006b) are a relatively new method of training multi-layer perceptrons. CB approaches heuristically seek to minimize classification error more directly by backpropagating distinct error signals from correctly classified and misclassified patterns, and from target and non-target output nodes. They tend to perform relatively minimal updates to network parameters in order to discourage premature weight saturation and overfitting. This is conducive to higher accuracy in classification problems than optimizing with respect to commonly used error functions, such as sum-squared error (SSE) and cross-entropy (CE).

We have observed a growing propensity in the machine learning community to reject the merit of new machine learning algorithms based solely on published empirical results in comparative testing. Several reasons for this are described in (Salzberg, 1999), including lack of real problem validation, testing on only a single or few data sets, parameter tuning with the help of test data, and experimental data being accompanied with insufficient statistical support. Given that existing classification-based error functions are heuristic in nature, intuitive and perform well in practice but with only modest theoretical foundation, the purpose of this work is to present a sufficient empirical and statistical comparison of CB and conventional error functions to assert general utility in classification domains.

Statistical significance results are shown across several isolated parameter controls that are known to influence convergence in backpropagation training: sensitivity to initial network parameter settings, variance in training patterns, pattern presentation order, learning rate, and number of hidden nodes across several benchmark applications. The purpose of these tests is to demonstrate algorithm robustness independent of parameter tuning on specific algorithms or datasets. Across all tests, the CB3 algorithm is shown to be predominantly superior, suggesting CB3 is capable of performing well while requiring a minimum of parameter tuning. Further behavioral analysis support these results and suggest promising avenues in further research into classification-based learning.

A background discussion of issues involved in selecting an objective function for neural network error backpropagation is provided in section 2. The CB1-3 algorithms are outlined in section 3. Experiments and results are presented in section 4. Further discussion and analysis is provided in section 5.

2 Conventional Objective Functions

To learn with gradient descent, an error function (also called a cost, loss or objective function) is applied to measure the deviance of model predictions from true values of problem instances. Although machine learning research has been mainly concerned with classification problems, gradient descent procedures, such as backpropagation, do not allow direct minimization of the number of misclassified training patterns (Duda, Hart, & Stork, 2001). Hence, the error function must be formulated such that classification

accuracy is increased as objective error is minimized. A problem with conventional error functions (i.e., SSE and CE), however, is that they do not always decrease monotonically with classification error (LeCun, Denker, & Solla, 1990). This, combined with other practical issues listed below, suggests that ANNs may be better suited to learning classification tasks by means of other error functions.

Learning to classify a pattern from N classes is often viewed as a regression problem with an N-valued response, with a target value of 1 in the n^{th} position if the observation falls in class n and 0 otherwise (LeBlanc & Tibshirani, 1993). The values of zero and one can be considered idealized, "true", or hard target values. However, ANNs have a real-valued output vector and are able to represent more general solutions than simple Boolean decisions. In practice, there is no reason why network solutions should require 0-1 values.

Using hard targets for training creates practical problems for ANNs. In order to output values approaching 0 and 1 (±1 for hyperbolic tangent), network weights must necessarily grow large. Also, since different portions of the problem tend to be learned at different times during training, using hard targets often leads to premature weight saturation, making it harder and slower to learn patterns that have yet to be learned (underfitting), while forcing the learner to overfit patterns that have already been learned. One common way to deal with this is to use "softer" targets like 0.1 and 0.9. This presents a less severe alternative but still suffers from these issues.

Rankprop (Caruana, Baluja, & Mitchell, 1996) provides an alternative method to training with 0-1 target values that exhibits empirical test improvement when measuring the area under the ROC curve. Rankprop records the output of the learner for each training pattern. It then sorts the patterns in the training set based on class, then according to output values. Thus, a rank of the patterns consistent with the current model is developed and used to define the target values on the next epoch. The idea behind Rankprop is that in the case of complex nonlinear solutions a simpler, less nonlinear function is learned instead.

The validity of using conventional differentiable measures like SSE as an objective function to minimize error relies on the assumption that pattern outputs are offset by inherent Gaussian noise, being normally distributed about a cluster mean. For approximating the function of an arbitrary signal this presumption often holds. However, this assumption is invalid for classification tasks where 0-1 target vectors are arbitrary values used to represent class labels.

Cross-entropy (CE) assumes idealized class outputs (i.e., target values of zero or one for a sigmoid activation) rather than noisy outputs as does SSE (Mitchell, 1997) and is therefore more appropriate to classification problems. CE is also preferable to SSE when the output class distributions are not balanced. However, by nature, CE also tends to weight saturation.

The classification figure-of-merit (CFM) objective function was introduced in (Hampshire II, 1990) for learning classification problems when it was shown that SSE and CE errors are not necessarily correlated with classification accuracy. CFM separates the values of network outputs by as large a range as possible such that error minimization is monotonic with increasing classification accuracy. Like SSE and CE, this metric encourages weight saturation, which is often indicative of overfitting and detrimental to generalization (Bartlett, 1998).

3 Classification-based error functions

Overfit is typically considered a global phenomenon. However, the degree of overfit can vary significantly throughout the input space. Caruana, Lawrence and Giles (2000) show that overly complex MLP models can improve the approximation in regions of underfitting, while not significantly overfitting in other regions. However, their discussion is limited to function approximation tasks and not classification problems, which are affected in a different way by bias-variance interactions (Friedman, 1997; Domingos, 2000).

A model's bias and variance, as defined in (Geman & Bienenstock, 1992), can be intuitively characterized as the model's error and its sensitivity to training data, respectively. Domingos (2000) formally defines bias as the loss incurred by a model's main prediction (the most common prediction for classifiers) relative to the optimal (Bayes) prediction and variance as the average loss of individual predictions relative to

its main prediction. Bias is independent of the training set and is zero for an optimal predictor. Variance is independent of test accuracy and is zero for a learner that does not take the training set into account when forming a hypothesis. There is an inherent tradeoff between fitting a limited training data sample perfectly and generalizing accurately on the entire population (Sharkey, 1996). Under these definitions, it is proven in (Friedman, 1997) that low squared-error bias is not important for classification, but rather 0-1 bias, and classification error may be reduced toward optimal by reducing variance alone (e.g., by using a cost function robust to idiosyncrasies in the training data).

The goal of training a neural network for classification is not to minimize the error between predetermined target and output values, but rather to produce output vectors that can be accurately translated to correct classifications. There are several possible ways to process the network's output vector in calculating an error signal for backpropagation to fit the data properly. A simple variant of using 0-1 targets involves augmenting the error function with a maximum error tolerance threshold, d_{max}, which is the smallest absolute output error to be backpropagated. In other words, given $d_{max} > 0$, a target value, t_j, and network output, o_j, no network update occurs if the absolute error $| t_j - o_j | < d_{max}$. This threshold is arbitrarily chosen to represent a point at which a pattern has been sufficiently approximated (usually 0.1, yielding target values of 0.1 and 0.9 instead of 0 and 1). With an error threshold, the network is permitted to converge with smaller weights (Schiffmann, Joost, & Werner, 1993). Weight decay (Krogh & Hertz, 1992) is a regularization approach that can discourage premature weight saturation and may improve generalization. More dynamic approaches, such as Rankprop (Caruana, 1995),

avoid the use of predefined hard targets, setting ranked soft target values for the training patterns each epoch. CB error functions are more dynamic, calculating soft targets online for each training pattern based on the network's current performance.

3.1 CB1 error function

Without loss of generality, it is assumed that a single, distinct output node in the network represents each class label. Let K be the number of output nodes in a network (and distinct class labels). Let o designate the activation value of a node ($0 \leq o \leq 1$ for sigmoid). Let o_k be the activation value of the k^{th} output node in the network. Let T designate the target class for the current training pattern and c_k signify the class label of the k^{th} output node. For target output nodes, $c_k = T$, and for non-target output nodes, $c_k \neq T$. Non-target output nodes are called *competitors*.

Let $o_{T\max}$ denote the maximum value among target output nodes (noting that there may be only one target node in many problem formulations),

$$o_{T\max} \equiv o_k : c_k = T.$$

Let $o_{\sim T\max}$ denote the value of the competitor outputting the highest o,

$$o_{\sim T\max} \equiv \max \{ o_k : c_k \neq T \}.$$

The error signal, ε, back-propagated from the k^{th} output node is defined as

$$\varepsilon_k \equiv \begin{cases} \min(o_{\sim T\max} + \mu - o_k, 1) & \text{if } c_k = T \text{ and } (o_{\sim T\max} + \mu \geq o_{T\max}) \\ \max(o_{T\max} - \mu - o_k, -1) & \text{if } c_k \neq T \text{ and } (o_k \geq o_{T\max} - \mu) \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where μ is a value between 0 and 1 that serves as an error or confidence margin between the outputs of target and competitor nodes, and min(·,1) and max(·,-1) enforce the [-1,1] range limit of the logistic function. Represented in closed form, the error (1) is

$$\varepsilon_k \equiv \min(o_{\sim T\max} + \mu - o_k, 1)\ I(c_k = T \text{ and } (o_{\sim T\max} + \mu \geq o_{T\max})) +$$
$$\max(o_{T\max} - \mu - o_k, -1)\ I(c_k \neq T \text{ and } (o_k \geq o_{T\max} - \mu)) \quad (2)$$

where I is the indicator or characteristic function. This is the error function to be minimized during training. Observe that when $o_{\sim T\max} + \mu < o_{T\max,}$ no error signal is backpropagated from the target output. Conversely, for a competing node k with output o_k, when $o_k < o_{T\max} - \mu$ no error signal is backpropagated from k. The error delta used for backpropagation is

$$\delta_k = \varepsilon_k f'(o_k)$$

where $f'(o_k)$ is the standard error gradient, which is

$$f'(o_k) = o_k(1 - o_k)$$

for a sigmoid squashing function, and can be removed on output nodes when using cross-entropy (Joost & Schiffmann, 1998). A more detailed discussion of CB1 is found in (Rimer & Martinez, 2006a).

3.2 CB2

CB2 (Rimer & Martinez, 2004) replaces the use of a single error function with a mixture taking advantage of both CB1 and SSE/CE optimization at appropriate times during the learning process. The heuristic is as follows:

> For each training epoch, let the error margin $\mu = t/T$, where $t \in \{0, 1, 2, \ldots\}$ is the current epoch and T is the maximum number of epochs to train.

CB2 causes a smooth transition from CB1 to SSE/CE optimization as the search progresses. The CB1 exploration phase first steers the decision surface toward a general problem solution without saturating network weights prematurely. Then, as learning tends toward SSE/CE exploitation, the distance of the decision boundary from proximate patterns is maximized. This approach is analogous to simulated annealing. CB2 has been shown to have smaller generalization variance on tested applications than CB1.

3.3 CB3

CB3 (Rimer & Martinez, 2006b) extends CB1 and CB2 by heuristically calculating how much error can be tolerated in each training pattern in order to improve generalization

based on how well the network is learning. That is, the error function itself is learned based on the network's ability to learn it. CB3 accomplishes this by observing how confident the network is in classifying each training pattern as learning progresses. This method is shown in Figure 1. CB3 augments predefined error functions in the following three ways:

- For each (pattern, output node) pair, a confidence value is stored and modified over time. This value represents a margin within the range of the squashing function that reflects the numeric amount by which the node is assisting in classifying the pattern correctly or incorrectly.
- As training progresses, target output values for each pattern are calculated using these learned confidence values.
- The error function is set by these target values. This decides how large an error signal to backpropagate.

In the initialization phase, confidence values are set for pattern-network output combined pairs. These values indicate the amount of confidence the network's output nodes expect to have in classifying the corresponding pattern correctly. Positive values semantically indicate a pattern is expected to be classified correctly while negative values mean it is expected to be misclassified. These values are updated as training progresses.

Initialization.
Set expected confidence values, $C_{i,n}$, for each pattern-output node pair.

Training.
Present a training pattern, i, to the network.
Determine $o_{T\max}$ and $o_{\sim T\max}$.

1. For each output node n, set its target output, T_n:

$$T_n \equiv \begin{cases} o_{\sim T\,\max} + (C_{i,n} + \alpha) & \text{if } c_n = T \text{ and } C_{i,n} > 0 \\ o_{T\,\max} - (C_{i,n} + \alpha) & \text{if } c_n \neq T \text{ and } C_{i,n} > 0 \\ o_{\sim T\,\max} & \text{if } c_n = T \text{ and } C_{i,n} \leq 0 \\ o_{T\,\max} & \text{if } c_n \neq T \text{ and } C_{i,n} \leq 0 \end{cases} \quad (1)$$

2. From each output node n, backpropagate error, ε_n.

$$\varepsilon_n \equiv \begin{cases} T_n - o_n & \text{if } c_n = T \text{ and } o_n < T_n \\ T_n - o_n & \text{if } c_n \neq T \text{ and } o_n > T_n \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

3. For all n, iteratively update confidence value $C_{i,n}$ to make it closer to the observed interval.

$$C_{i,n} = C_{i,n} + \eta_C\,(OI_n - C_{i,n}) \quad (3)$$

where η_C is a small (~0.01) confidence learning rate and the observed interval, OI_n, is defined as

$$OI_n \equiv \begin{cases} o_n - o_{\sim T\,\max} & \text{if } c_n = T \\ o_{T\,\max} - o_n & \text{if } c_n \neq T \end{cases} \quad (4)$$

Continue training until stopping criterion is satisfied.

Figure 1. CB3 algorithm.

For each training pattern, its vector of target values is determined (Figure 1, equation 1). Each output node's target value is calculated distinctly depending on whether it represents the target class and whether the confidence value for that node is positive or negative. The value α determines how aggressively CB3 will try to separate target values

for opposing classes. A value of zero will allow targets to remain close together while a greater value will separate them more.

Based on the calculated target vector, the error signal for backpropagation through the network is determined (Figure 1, equation 2). Observe that no error is backpropagated from an output node when its value is already closer to the corresponding limit of the squashing function than the calculated target.

Following backpropagation, the meta-learning step of iteratively updating learned confidences used in calculating the dynamic target values (Figure 1, equation 3) is performed. First, the observed interval (the difference between output node values on positive and negative class labels) vector is measured (Figure 1, equation 4), respective to each output node. Then, as an error delta is calculated for iterative weight updating to bring output values closer to the target values, the learned confidence value $C_{i,n}$ is subtracted from the observed interval OI_n and multiplied by small, positive learning rate η_C to calculate a confidence value delta. This confidence delta is added to the learned confidence.

As training continues, $C_{i,n}$ will be iteratively learned. It reflects the ability of the network output nodes to locally distinguish the target class from the others. Learning this value has the practical purpose of hinting to the network which patterns can be classified with confidence (such as cluster centers), which patterns need to be learned more (fringe patterns), and which patterns appear difficult to learn correctly with the current network

at all (noise or overlapping classes). This enables the training process to guide the network to spend more resources on learning training patterns that most probably contribute to higher generalization accuracy while selectively ignoring those that lead the network to overfit or weight saturation as it attempts to learn them.

The CB3 error function introduces three new learning parameters: $C_{i,n}$, the confidence pattern-class vector, η_C, a confidence learning rate, and α, a confidence multiplier used in separating competing output node target values. In all experiments presented here, $C_{i,n}$ = 0.25, η_C = 0.01, and α = 1.5. These values were shown to perform uniformly well across several test applications (Rimer & Martinez, 2006b).

4 Experiments

Experiments were performed comparing six error functions: sum-squared error (SSE), cross-entropy (CE), CE WD (cross-entropy with weight decay of λ = 0.00001 (Krogh & Hertz, 1992)), CB1, CB2, and CB3. Eleven benchmark applications from the UC Irvine Machine Learning Repository (UCI MLR) (Blake & Merz, 1998) were used in testing (see Table 1). The problems were selected so as to have a wide variety of characteristics (size, number and type of features, complexity) in order to analyze the robustness of these error functions to varying learning conditions. Feature values were normalized between zero and one.

To learn each problem, a fully connected feed-forward network with one output node per class label and a single hidden layer trained through online backpropagation was used. Training patterns were randomly shuffled before each epoch. An error tolerance threshold (d_{max}, described in section 3) of 0.1 was used. Pattern classification was determined by *winner-take-all* (the class of the highest outputting node is chosen) on all models tested. Training continued until the training set was successfully learned or until a decrease in classification error on a holdout set was not observed for 500 consecutive epochs. The model selected for testing was the one with the best holdout set classification accuracy. Results were collected using 10-fold stratified cross-validation.

4.1 Training parameters

Four network learning or model parameters were used as control variables:

- initial network weight values
- pattern presentation order
- learning rate, and
- number of hidden nodes.

These parameters have been shown to be influential to multilayer perceptron learning over repeated studies in the literature. Numerous techniques of setting these learning controls have been put forth, including heuristic rule-of-thumb approaches, tuning by empirical testing, or based on more sophisticated algorithms (Campbell & Coombes, 1995; Thimm and Fiesler, 1997; Istook & Martinez, 2002). Rather than show how the error functions perform under specific conditions with optimized parameter values, our goal is to present their robustness across the entire practical range of parameter values

(see sections 4.2 to 4.5). In other words, we were interested in how each condition affects the performance of each error function, and also whether any error function dominates (outperforms the other error functions across all parameter values).

Except where noted in each experiment in this section, learning parameter controls were fixed as follows. Network weights were initialized to uniform random values in the range [-0.1, 0.1] (Thimm and Fiesler, 1997). Pattern presentation order was set by always shuffling the order using the same random seed. Learning rate was 0.1 and momentum was 0.7. The number of hidden nodes used was selected by determining the minimum size required to achieve near (within 0.5%) the best holdout set accuracy observed with each of these error functions (discussed further in section 4.5). The datasets, number of patterns in the set, and network architecture used for each dataset are listed in Table 1.

Table 1. Datasets and network architectures.

Dataset	# Patterns	Network
ann	7200	21-30-3
balance	625	4-6-3
bcw	699	9-10-2
derm	358	34-5-6
ecoli	336	7-8-8
ionosphere	351	33-9-2
iris	150	4-2-3
musk2	6598	166-5-2
pima	768	8-9-2
sonar	208	60-15-2
wine	178	13-16-3

4.2 Effect of variance in initial network weights

First, we evaluated the influence of variance in initial network weights on test accuracy. We performed ten-fold stratified cross-validation thirty times on each data set, each time with a distinct random seed used in initializing network weights. That is, one seed value was used for all tests on the first run, a second seed used for all tests on the second run, etc.

Figure 2 shows aggregated test accuracies and 95% confidence intervals for these error functions by column. Results for a support-vector machine using the LIBSVM software library with optimal parameter search (Chang & Lin, 2001) are included in the first column for baseline comparison (without the network confidence intervals shown for variance in initial parameter settings).

The neural network models performed significantly better than an optimal SVM on these datasets. Varying only initial weight parameters, CB1-3 and CE performed significantly better than SSE. CB1 and CB2 performed better than CE on average, but not significantly. CB2 has the tightest confidence interval, indicating that it is most robust to variance in initial weight values. CB3 has significantly better test accuracy than all other algorithms. This would indicate that CB3 performs the best, and CB1-3 and CE will outperform SSE, given the specified learning parameters, for any reasonable sets of initial network weights in the distribution we used.

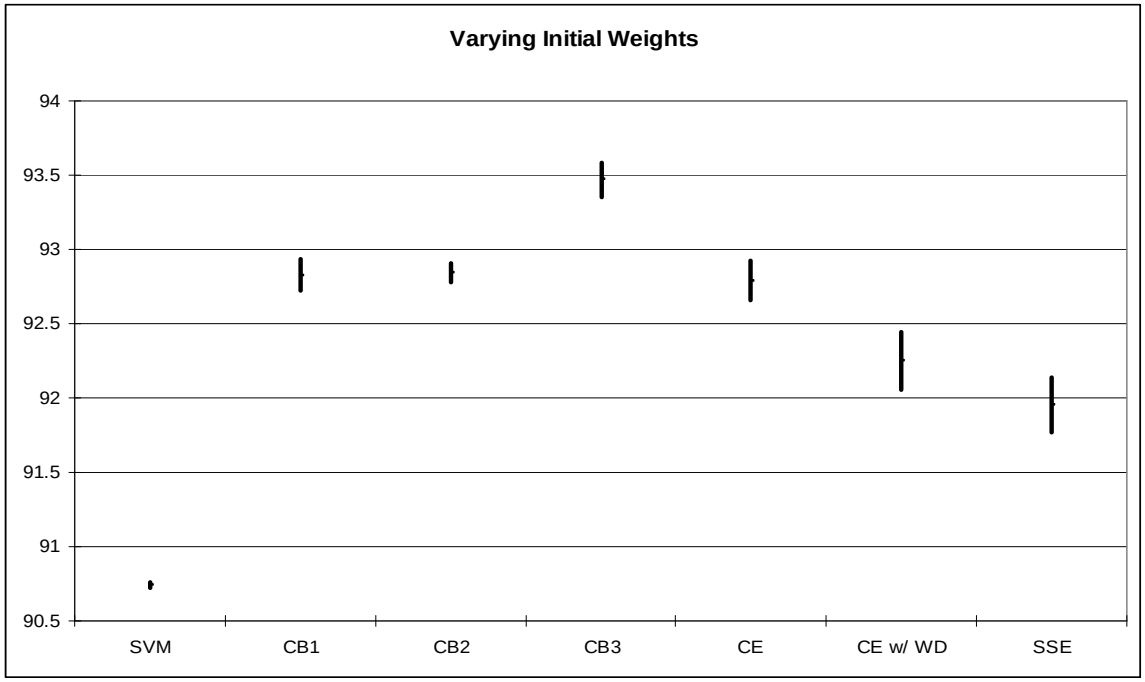

Figure 2. Test performance averaged over thirty runs with different initial weight settings.

4.3 Effect of variance in pattern presentation order

Next, we determined the influence of varying pattern presentation order. Each application was tested thirty times, with seed values of 1 to 30 used to shuffle the patterns. That is, one seed value was used for all tests on the first run, a second seed used for all tests on the second run, etc. The results of this test are more applicable to real-world ANN usage than the study in section 4.2 because the effect pattern presentation order has subsumes the effect of initializing the network to any sufficiently small random weights. An intuitive reason for this is that the error backpropagated due to presenting a certain training pattern (i.e., input vector) generally results in a distinct set of weights than from presenting any other pattern. Hence, which training patterns are stochastically presented first to the network influences the direction of updates more than how the

initial weight values given affects subsequent update direction. Naturally, this logic does not apply to batch training, but does apply to mini-batch approaches.

The same initial network weight values were used for each algorithm on a given dataset. All other parameters were set as before. The aggregated test accuracies with 95% confidence intervals are shown in Figure 3.

CB1-3 and CE performed significantly better than CE WD and SSE. CB2 performed better than CB1, and CB1 better than CE, but not at 95% significance. CB2 is significantly better than CE, and CB3 exhibits the highest test accuracy by a significant margin. CB3 also has the tightest confidence interval, followed by CB2, indicating these error functions are most robust to pattern presentation order. This indicates that, given the specified network architectures and learning rate, CB3 outperforms the other error functions on these datasets for reasonable sets of initial network weights and pattern distributions. Consistent with the notion that pattern presentation order has a greater influence on the final state of the network than does the initial network state, it may be observed that average test error is lower and confidence intervals are wider in Figure 3 than in Figure 2.

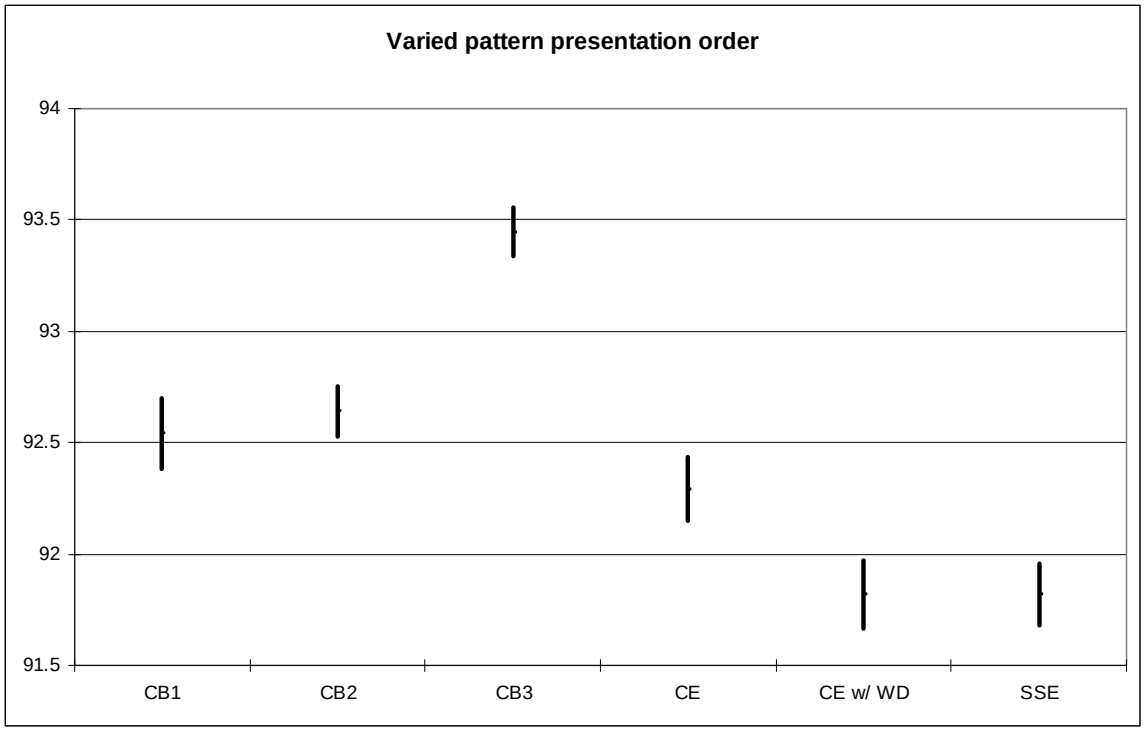

Figure 3. Test performance averaged over thirty runs with different pattern presentation orders.

Table 2 lists the standard deviation in test accuracy (in percent) due to pattern variance across the training set partitions used in 10-fold cross-validation, averaged over all test runs performed in this section, plus or minus a 95% confidence interval. Observe these values are much greater than deviation in accuracy due to initial network weights or pattern presentation order. Squared-error cost functions exhibit the highest model variance and CB3 exhibits the lowest variance.

Table 2. Aggregate standard deviation in test accuracy (in percent) due to pattern variance.

CB1	CB2	CB3	CE	CE WD	SSE
5.47±1.02	5.42±1.01	5.04±0.94	5.60±1.05	5.62±1.05	5.66±1.06

4.4 Effect of varying the learning rate

Third, the influence of learning rate was evaluated. We re-ran the tests thirty times, each time using a learning rate from 0.01 to 0.3 in uniform increments. Generalization accuracy, averaged over the applications tested, is shown for each error function and learning rate in Figure 4. Confidence intervals are not shown here for clarity, but remained near the ranges shown in Figure 3.

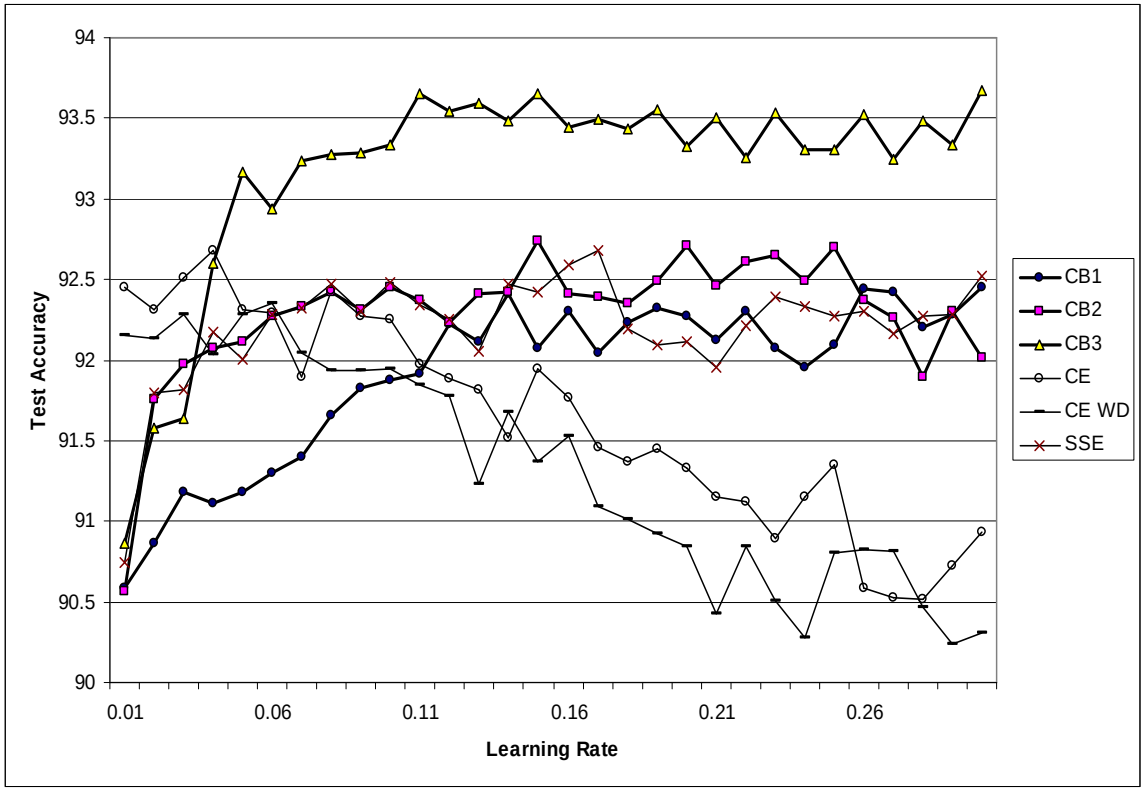

Figure 4. Average test accuracy over applications at various learning rates.

CB1's performance is less at a learning rate below 0.1 and roughly steady above 0.1. CB2 performs best with a learning rate between 0.05 and 0.25. CB3 performs best at a learning rate of 0.1 or higher. SSE performed roughly the same for learning rates above 0.05 and worse for learning rates below 0.05. However, CE and CE WD performed best

for learning rates of 0.1 or less and their performance degraded steadily for greater learning rates. We surmise this is due to having a much higher error signal backpropagated through the network for a given target-output difference than any of the other error functions, making it difficult to converge with larger step sizes. While the other algorithms perform roughly the same for these applications and random seed, given their *optimal* learning rate, CB3 is shown to always perform significantly better than the other error functions for *any* learning rate above 0.05. This is useful to know, for it indicates that CB3 performs well without learning rate tuning for specific applications. The possibility of removing the backpropagation learning rate parameter entirely from CB3 training is slated for future work.

4.5 Effect of varying the number of hidden nodes

It is sometimes believed that networks with too many degrees of freedom generalize poorly. This line of reasoning is based on two observations: (1) that a sufficiently large network is able to memorize the training data if training continues long enough, and (2) even with early stopping approaches, it is not apparent whether some form of overfit has occurred. By reducing the learning capacity of such a network, it is thereby forced to generalize as it no longer has the capability to memorize the training data. In order to perform a proper theoretical analysis of network capacity and generalization, the search heuristic must also be taken into account (Caruana, 1997; Caruana, Lawrence & Giles, 2000). Gradient descent search heuristics do not give all hypotheses an equal opportunity. The inductive bias of standard backpropagation is to start with a simple hypothesis (through usually small, random weights) and make the hypothesis more

complex (by increasing the magnitude of the weights) until the network sufficiently learns the problem. Thus, backpropagation is biased toward hypotheses with small weights, examining solutions with larger weights only as dictated by necessity. Excess network capacity does not necessarily hinder generalization, as learning stops as soon as possible. This means that generalization can be less sensitive to excess network capacity, and that using a network that is too small can hurt generalization more than using networks that are too large (Caruana, 1997).

To verify this, we tested how well each error function performs over a range of network sizes. For this experiment, we reran the above experiment thirty times, using networks where the number of nodes in the hidden layer was varied from 1 to 30 nodes. Generalization, averaged over the applications tested, is shown for each error function and number of hidden nodes in Figure 5.

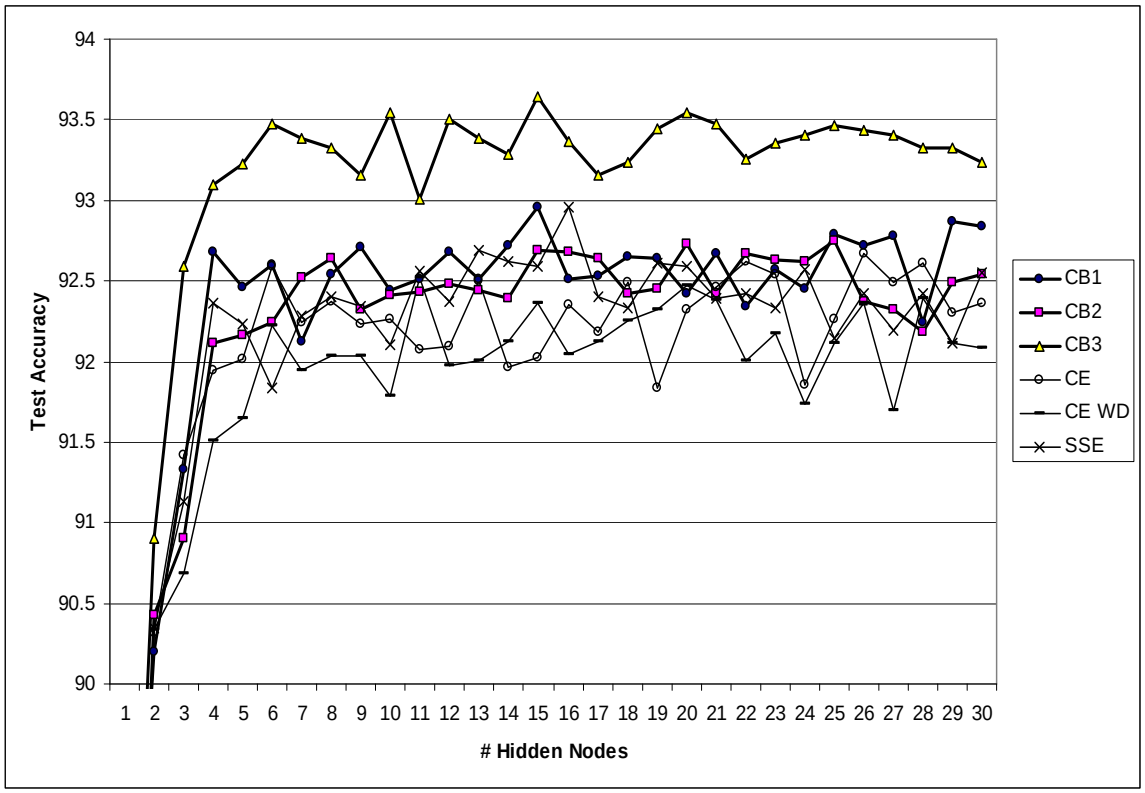

Figure 5. Average test accuracy over applications at various learning rates.

For this application corpus, all error functions approximated their best demonstrated test accuracy with six hidden nodes or higher. This is in keeping with observations that proper early stopping heuristics are more important to network generalization than the number of hidden nodes in a sufficiently large network (Caruana, Lawrence & Giles, 2000) and that stopping learning before the global error minimum has the effect of network size selection (Wang, Venkatesh, & Judd, 1994). However, CB3 performed significantly better than the other error functions across all networks of matching size. Also, CB3 networks of four or more hidden nodes performed better than networks trained using the other error functions of *any* size.

The results of sections 4.2-4.5 indicate that CB3 is a better classification learner, independent of network size, learning rate, and idiosyncrasies in initial network conditions and pattern presentation order.

5 Analysis

It has been shown that the number of nodes in a network is not as influential as the magnitude of the weights (Bartlett, 1998). The topology, rather, serves more as a mechanism that lends itself to solving of certain problems, while the weights represent how tightly the network has fit itself to the (admittedly incomplete) training data distribution. Network complexity is defined in (Wang, Venkatesh, & Judd, 1994) as the number of parameters and the capacity to which they are used in learning (i.e., their magnitude). They show it is best to make minimal use of the capacity of the network for encoding the information provided by the learning patterns, which is in keeping with the methodology of CB error functions.

5.1 Network parameter sizes

Problems of premature weight saturation, such as underfitting and increased model bias, were discussed in section 2. Here we examine to what extent the error functions we tested encourage weight saturation. Using the network sizes and learning parameter values listed in section 4.2, we performed a trace of network weight parameter magnitudes during training. We observed that CB3 retains the smallest weight magnitudes during the entire training process, followed in order by CB1, CB2, SSE, CE

WD, and CE, which induces the largest network parameters. For brevity, we present histograms of network weight magnitudes while learning the *bcw* dataset only (see Figures 6a-f), noting that this relative ordering was observed across all tested applications. On each graph, note that at epoch 0, all weights are almost 0. As training progresses, these weights tend to become larger. Figure 7 is a condensed version of these, showing the mean weight size during training for each error function.

Generally, weight magnitudes of 1.0 or less are considered to be in the approximately linear region of the sigmoid squashing function, while parameters become saturated (i.e., enter the asymptotic region of the sigmoid) with values above 2.0. Figure 6a demonstrates how CB1 avoids saturating weights (with the exception of a single bias weight in the situation shown here). CB2's behavior (Figure 6b) is very similar to that of CB1. It can be observed how the steadily increasing value for μ (recall its use as a margin booster, mentioned in section 3.2) makes small weights larger at a slow, even pace. Using CB3 (Figure 6c), most network weights remain in the linear region of the sigmoid. The smoothness of CB3's weight histogram across training reflects the intuition that CB3 alters pattern target values methodically, without straining the network's capacity. It is conceivable that CB3 will also behave well using simpler linear output activations without further algorithm modification or tuning of training parameters.

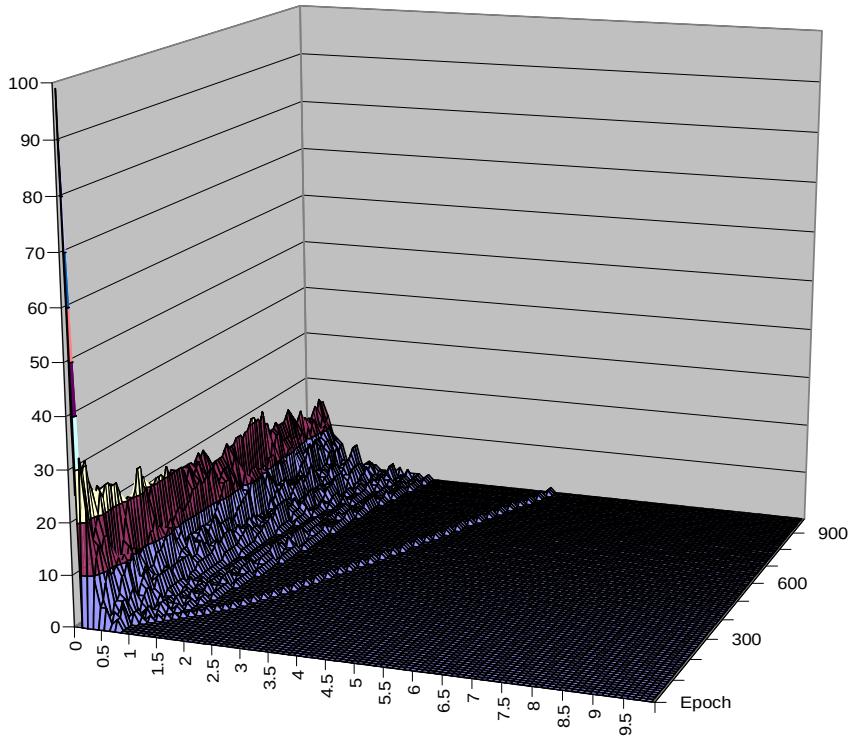

Figure 6a. Weight magnitudes as training progresses with CB1.

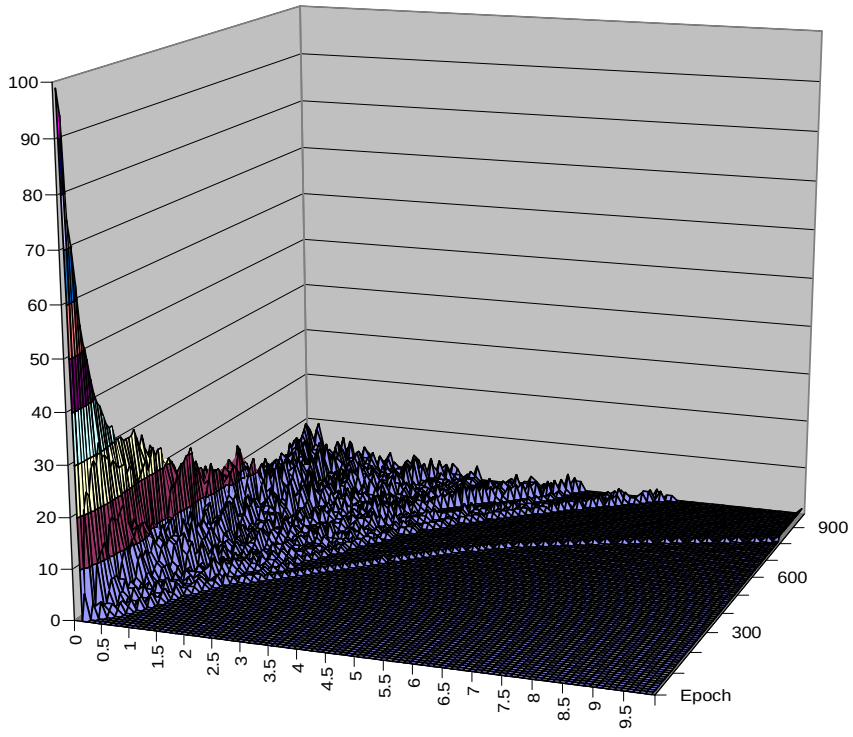

Figure 6b. Weight magnitudes as training progresses with CB2.

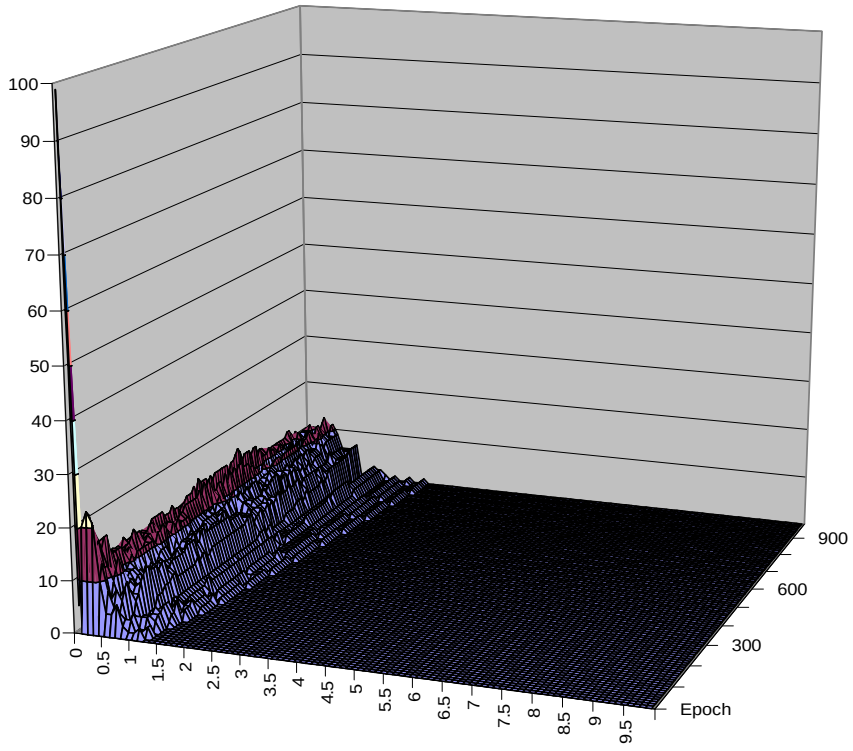

Figure 6c. Weight magnitudes as training progresses with CB3.

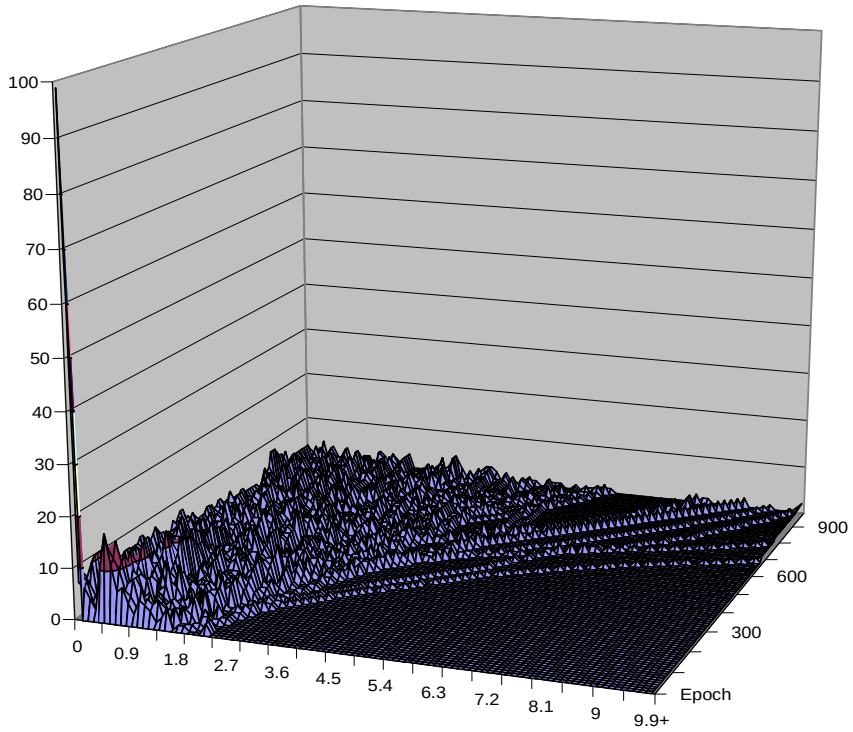

Figure 6d. Weight magnitudes as training progresses with SSE.

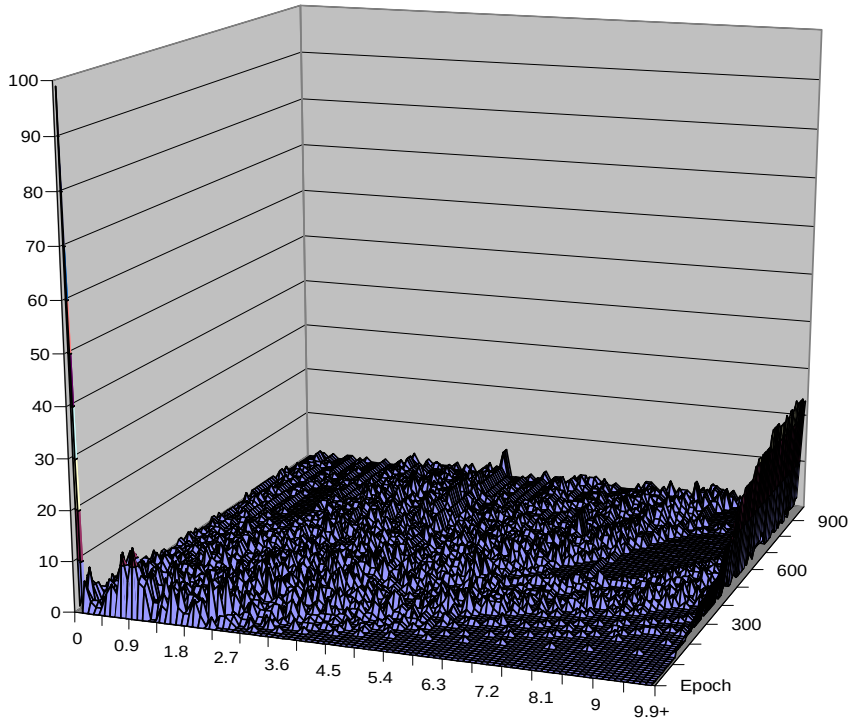

Figure 6e. Weight magnitudes as training progresses with CE.

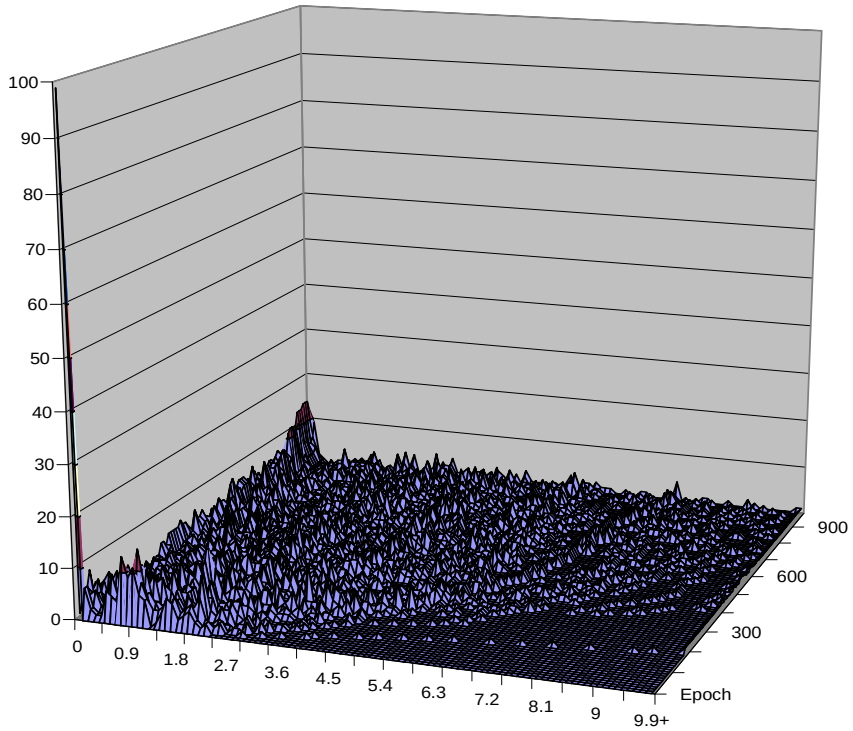

Figure 6f. Weight magnitudes as training progresses with CE with weight decay.

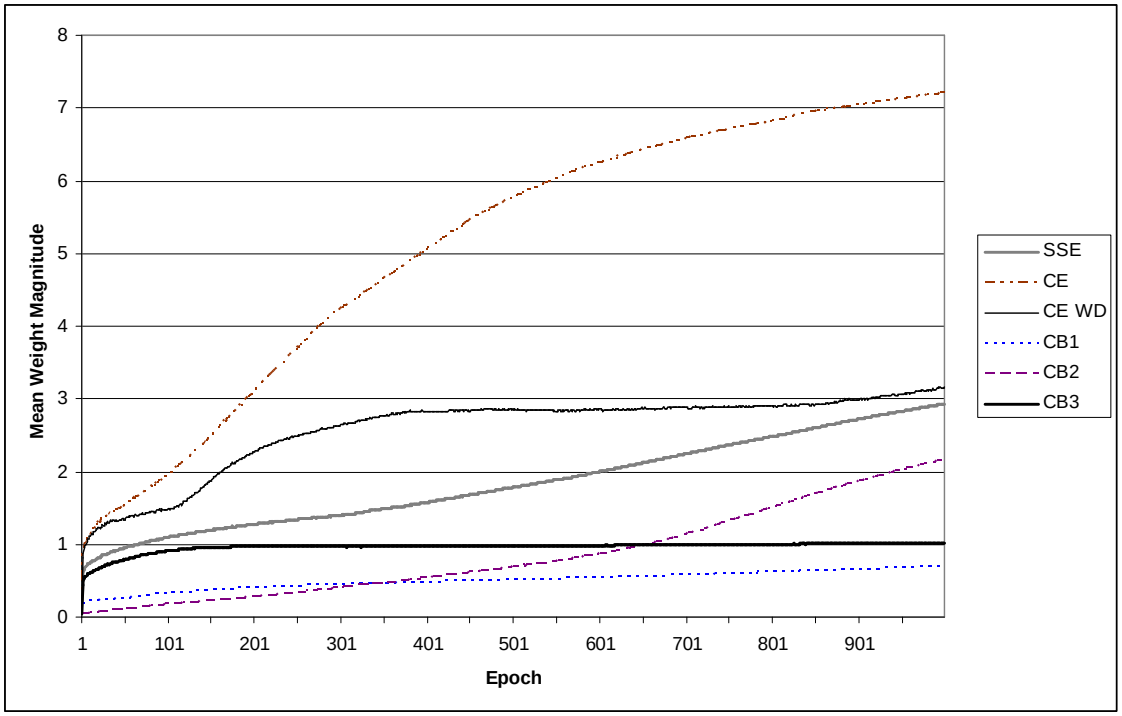

Figure 7. Mean weight magnitude as training progresses on *bcw*.

SSE (Figure 6d) produces larger average weights than CB1-3, and it can be observed how network parameters saturate steadily from the first training epochs. CE (Figure 6e) has the greatest tendency to weight saturation because it is defined to calculate the greatest error signal of all these error functions for a given learning rate. Hence, well behaved learning rates tend to be less for CE than for SSE and CB1-3. A smaller learning rate provides smoother convergence and slows weight saturation. However, these are often achieved at the cost of a greater number of training epochs before stopping (i.e., the epoch from which the final/best network model is selected). We have observed that the models with the highest holdout set accuracy have roughly the same average weight magnitude, regardless of (reasonable) learning rate. Therefore, it is difficult to say whether small learning rates provide any practical improvement for SSE or CE with respect to avoiding *premature* weight saturation. Weight decay, however, does

demonstrably ameliorate the saturation of weights (Figure 6f), although the phenomenon of saturation is still readily observed.

5.2 Classifier output difference

Before training, networks initialized with small, random weights typically have high mean squared-error. It has been shown that high accuracy network solutions exist where measured squared-error is nearly as high as a network comprised of small, randomly initialized weights (Rimer & Martinez, 2006a). This evidence, coupled with the radically distinct weight distributions observed above, lend credence to the notion that CB error functions induce a fundamentally different search of hypothesis space than squared-error or cross-entropy optimization.

We explore this further by calculating the classifier output difference (COD) distance described in (Peterson & Martinez, 2005) among the network solutions from the experiments performed above. The COD distance between two hypotheses is the frequency (a real value between 0 and 1) that they disagree on pattern classification. This distance can be estimated by observing the frequency that the hypotheses, H_1 and H_2, disagree with each other on the classification of test patterns:

$$\hat{D}_T(H_1, H_2) = \frac{\sum_{x \in T} I(H_1(x) \neq H_2(x))}{|T|}$$

where T is the set of test patterns and I is the indicator or characteristic function.

We calculated aggregate COD distances for the hypotheses selected for the experiments in section 4.3, both within the group of hypotheses induced by an error function, which we will call *intra-class* comparisons, and also against the hypotheses of the other error functions, or *inter-class* comparison. Table 3 lists the COD distances, averaged across all applications and test runs. Intra-class distances are shown along the diagonal and inter-class distances are shown in non-diagonal cells. This matrix is symmetric, so the values in lower triangle are omitted here for clarity. Average inter-class COD for each error function and the difference of this value from the intra-class COD ("distinction") are shown in the bottom two rows.

Table 3. COD distances.

Error function	CB1	CB2	CB3	CE	CE WD	SSE
CB1	**.0282**	.0368[2]	.0431	.0435	.0449	.0494
CB2		**.0302**	.0404	.0408	.0439	.0429
CB3			**.0260**	.0461	.0493	.0483
CE				**.0330**	.0354[1]	.0417
CE WD					**.0338**	.0442
SSE						**.0344**
average	.0435	.0409	.0455	.0415	.0436	.0453
distinction	.0153	.0107	.0195	.0085	.0098	.0109

Similar error functions have low COD when compared to one another, and vice-versa. Here, CE and CE WD have the lowest distance from one another (cell denoted by the superscript '1'), followed by CB1 and CB2 (denoted by the superscript '2'), which is as we would expect due to their similar natures.

For these models (from section 4.3), intra-class COD shows behavioral test difference due to randomness in pattern presentation of otherwise identical training situations.

Smaller intra-class COD is suggestive of less variance in the error function. The relative ordering of values along the diagonal is generally consistent with the relative sizes of confidence intervals shown in Figure 3. In both cases, CB3 has the smallest variance, showing it to be the most robust of these error functions in this regard.

The difference between intra- and inter-class distances (shown in the bottom row of Table 3) translates to distinction in the particular error function's behavior. A distinction of zero indicates the behavioral difference of this error function from the others is on average functionally indistinguishable from intra-class model variance due to pattern presentation order. Positive non-zero values indicate that the error functions are measurably distinct from the others. CB3, followed by CB1, show the greatest level of behavioral distinction from the other error functions.

Table 3 may be depicted in three-dimensions by calculating a best fit reduction of the five-dimensional (non-Euclidean) COD distances among these error functions (see Figure 8). In general, the sphere volumes represent intra-class COD on the considered applications and may be interpreted as approximate representations of the variance of solutions that networks using a particular error function converge to, given the training parameter settings. Inter-class COD is depicted by the distance between centers. The farther apart two spheres are, the more often hypotheses of the two error functions disagree. Observe the CB error functions are behaviorally closer to one another than to the squared-error functions. CB3 has the smallest sphere, illustrating it has the lowest model variance.

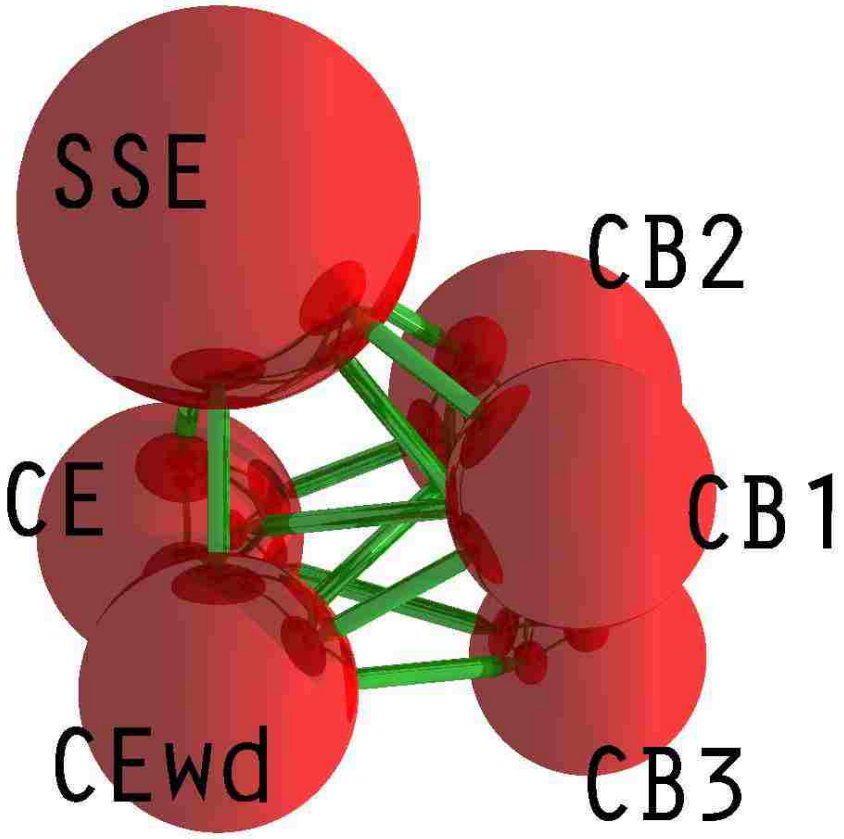

Figure 8. COD distances of Table 3 in three dimensions.

These error functions' 95% confidence intervals cover a range of 0.2-0.3% (see Figure 3). However, their average intra-class COD distances range from about 2.6% to 3.4% (see Table 3). This indicates much greater behavioral variance across solutions than is observable solely through observing variance in test accuracy. Furthermore, while difference in test accuracy across error functions is about 1-2%, Table 3 shows average behavioral difference across these error functions to be greater than 4%. With test error averaging about 7-8% (see Figure 3), this indicates a low correlation of classification errors across error functions. It is therefore likely that combining models induced by different error functions into hybrid ensembles or voting committees would produce

higher test accuracy than that of homogeneous ensembles. This will be considered in future work.

6 Conclusion

The CB1-3 error functions generalize significantly better than squared-error minimization over the tested classification applications on average. CB3 generalizes significantly better than the other error functions tested and is most robust to pattern variance, initial network weight values, pattern presentation order, learning rate, and number of hidden nodes, suggesting it operates consistently well with minimal or no parameter tuning or operator intervention in the training process.

Observing the magnitude of network weight parameters during training showed CB error functions avoid premature weight saturation on the problems tested. It was also shown that classification errors between networks trained with different error functions have relatively low correlation. It is expected that hybrid ensembles or voting committees of CB, SSE, and CE error functions will reduce test error more than homogeneous ensembles of networks trained using any of these error functions.

Acknowledgments: We thank Adam Peterson for the use of his 3-D rendering tool.

References

Bartlett, P. (1998). The Sample Complexity of Pattern Classification with Neural Networks: The Size of the Weights is More Important than the Size of the Network. *IEEE Trans. Inf. Theory, 44:2*, 525-536.

Blake, C. & Merz, C. (1998). UCI Machine Learning Repository, http://www.ics.uci.edu/~mlearn/MLRepository.html. Irvine, CA: University of California, Department of Information and Computer Science.

Campbell, C. & Coombes. S. (1995). Determining the Optimal Number of Hidden Nodes in a Feed-Forward Neural Network. Dept. of Engineering Mathematics, Bristol University.

Caruana, R. (1995). Learning Many Related Tasks at the Same Time With Backpropagation. *Advances in Neural Information Processing Systems* (NIPS) *7*, pp. 657-664.

Caruana, R., Baluja, S. & Mitchell, T. (1996). Using the Future to 'Sort Out' the Present: Rankprop and Multitask Learning for Medical Risk Evaluation. *Advances in Neural Information Processing Systems* (NIPS) *8*.

Caruana, R. (1997). *Multitask Learning*. Ph.D. Thesis, School of Computer Science, CMU.

Caruana, R., Lawrence, S. & Giles, C. (2000). Overfitting in Neural Networks: Backpropagation, Conjugate Gradient, and Early Stopping. Proceedings of *Neural Information Processing Systems* (NIPS) *13*.

Caruana, R. & Niculescu-Mizil, A. (2006). An Empirical Comparison of Supervised Learning Algorithms. Proceedings of the *23rd International Conference on Machine Learning*.

Chang, C. & Lin, C. (2001). LIBSVM: a library for support vector machines. http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Domingos, P. (2000). A Unified Bias-Variance Decomposition for Zero-One and Squared Loss. Proceedings of the *17th National Conference on Artificial Intelligence*.

Duda, R., Hart, P. & Stork, D. (2001). *Pattern Classification*, 2nd edition. John Wiley & Sons, Inc.

Friedman, J. (1997). On Bias, Variance, 0/1-Loss, and the Curse-of-Dimensionality. *Data Mining and Knowledge Discovery, 1:1*, 55-77. Kluwer Academic Publishers.

Geman, S. & Bienenstock, E. (1992). Neural Networks and the Bias/Variance Dilemma. *Neural Computation, 4*, 1-58.

Hampshire II, J. (1990). A Novel Objective Function for Improved Phoneme Recognition Using Time-Delay Neural Networks. *IEEE Transactions on Neural Networks, 1*:*2*.

Istook, E. & Martinez, T. (2002). Improved Backpropagation Learning in Neural Networks with Windowed Momentum. International Journal of Neural Systems, 12(3-4), 303-318.

Joost, M. & Schiffmann, W. (1998). Speeding up Backpropagation Algorithms by using Cross--Entropy combined with Pattern Normalization. *International Journal of Uncertainity, Fuzziness and Knowledge-based Systems (IJUFKS), 6*:*2*, 117-126.

Krogh, A. & Hertz, J. (1992). A Simple Weight Decay Can Improve Generalization. *Advances in Neural Information Processing Systems* (NIPS) *4*, San Mateo, CA, 950-957.

LeBlanc, M. & Tibshirani, R. (1993). Combining estimates in regression and classification. *NeuroProse*.

LeCun, Y., Denker, J. & Solla, S. (1990). Optimal Brain Damage. In Touretzky, D. (ed.), *Advances in Neural Information Processing Systems (NIPS) 2*, 598-605. San Mateo, CA: Morgan Kaufmann.

Mitchell, T. (1997). *Machine Learning.* McGraw-Hill Companies, Inc., Boston.

Peterson, Adam H. and Martinez, Tony R. (2005). Estimating the potential for combining learning models. Proceedings of the *ICML Workshop on Meta-Learning*, 68-75.

Rimer, M. & Martinez, T. (2004). Softprop: Softmax Neural Network Backpropagation Learning. Proceedings of the *IEEE International Joint Conference on Neural Networks IJCNN'04*, 979-984.

Rimer, M. & Martinez T. (2006a). Classification-based Objective Functions. *Machine Learning*, *63:2*, 183-205.

Rimer, M. & Martinez T. (2006b). CB3: An Adaptive Error Function for Backpropagation Training. *Neural Processing Letters*, 24:1, 81-92.

Salzberg, S. (1999). On Comparing Classifiers: A Critique of Current Research and Methods. *Data Mining and Knowledge Discovery*, *1*, 1-12.

Schiffmann, W., Joost, M., & Werner, R. (1993). Comparison of Optimized Backpropagation Algorithms. *Artificial Neural Networks*. European Symposium, Brussels.

Sharkey, A. (1996). On Combining Artificial Neural Nets. *Connection Science, 8:3-4*, 299-313.

Thimm, G. & Fiesler, E. (1997). High-order and Multilayer Perceptron Initialization. *IEEE Transactions on Neural Networks*, 8:2, 249-259.

Wang, C., Venkatesh, S., & Judd, J. (1994). Optimal stopping and effective machine complexity in learning. In Cowan, J., Tesauro, G., & Alspector, J. (eds.), *Advances in Neural Information Processing Systems (NIPS) 6*, 303-310. Morgan Kaufmann, San Francisco.

Chapter 7

Improving Posteriors with Point-wise Local Binning

Michael Rimer, Adam Peterson and Tony Martinez
Department of Computer Science, Brigham Young University, Provo, UT 84602

Abstract. Recent work has shown the efficacy of calibrating learning model outputs to provide more accurate probability predictions. We present a novel point-wise local binning method, PL1, for calibrating model outputs and measuring model calibration fitness. This method is compared to modern methods for model calibration: Platt Scaling and Isotonic Regression. While these methods have been shown previously to not improve neural network calibration fitness or classification accuracy, we show that PL1 does significantly reduce neural network calibration error to improve general probability predictions. We also quantitatively compare PL1 to isotonic regression on naïve Bayes, *k*-nearest neighbor, and bagged decision tree ensembles, for which they perform comparably.

1 Introduction

In many machine learning applications, it is important for the learning model to yield accurate posteriors, for instance, when the cost function is asymmetric or unknown, or where the outputs of one layer of a multi-tier system are used as probabilities by another. In these situations, achieving high classification accuracy may not be sufficient. Certain learning methods output values that may be used naturally as posteriors, such as softmaxed neural networks and bagged decision tree ensembles. Other learning models

do not output posteriors, such as decision trees, or may output biased posteriors (e.g. naïve Bayes and SVMs) [1].

Reliability diagrams [2] are commonly used in visualizing model calibration for real problem domains where true conditional probabilities are not known. The accepted method for generating these diagrams involves discretizing the prediction space into a set number of exclusive bins and plotting the mean predicted value against the true (i.e., empirically measured) fraction of positive cases. For a well-calibrated model, predictions will fall along the diagonal line, representing a model whose confidence matches the actual posterior probability for a given data set (see Figure 1). Calibration mean-squared error (CMSE) is calculated by averaging the squared distance of the model's prediction from the measured posterior across all bins.

Another way of calculating reliability is by using a proper scoring rule, which has the properties of maximizing score when the model outputs correct probabilities, while also discriminating optimally with respect to classification. One such example of a proper scoring rule is sum-squared error, also called the Brier score [3].

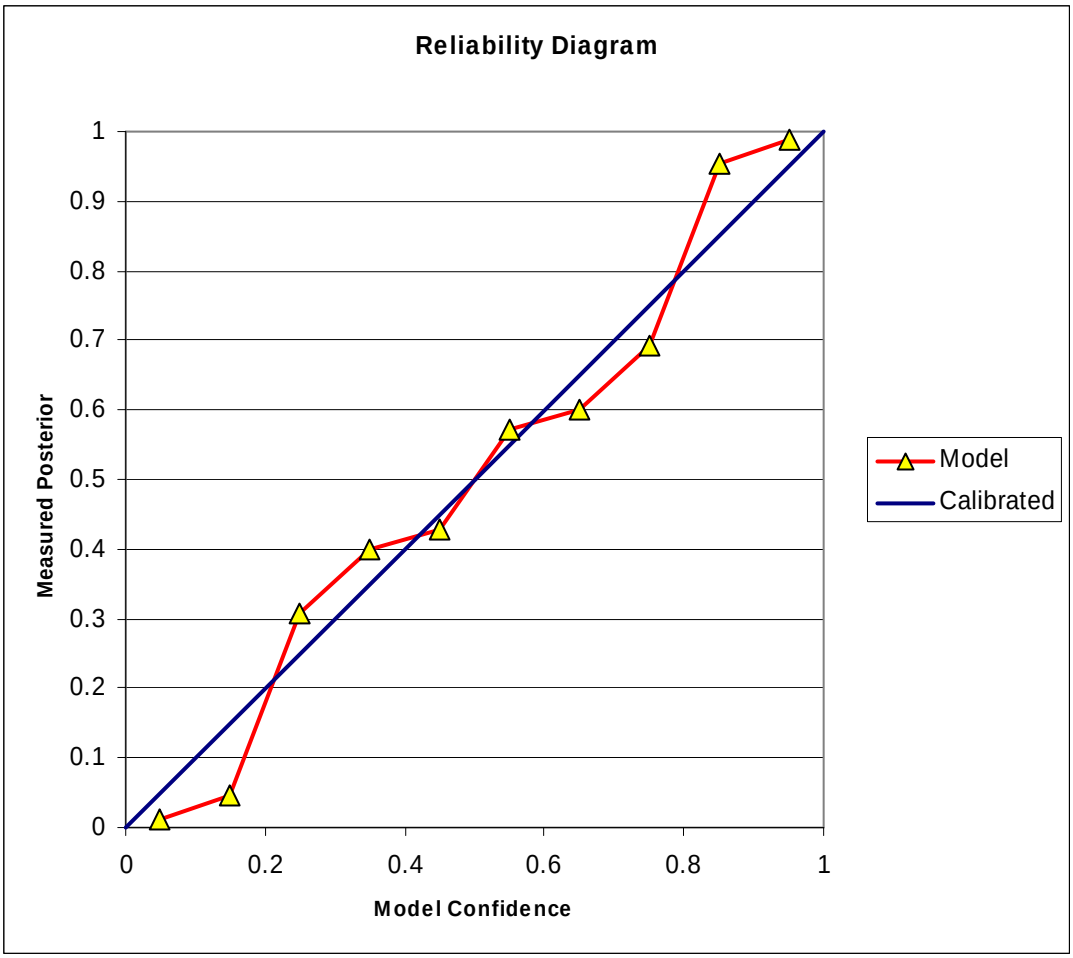

Figure 1. Reliability diagram using the conventional 10-bin method.

In certain cases, introducing a post-processing step of model output calibration, such as Platt Scaling [4] or Isotonic Regression [5,6], may yield more accurate or less biased posteriors. Niculescu-Mizil and Caruana [1] showed that using reliability diagrams, these methods may be effective in improving model calibration for boosted trees, random forests, and SVMs, while not reducing or increasing calibration error for neural networks.

In this work, we propose a point-wise local binning method, PL1, for performing model calibration and measuring reliability. PL1 has intuitive and practical improvements over

the current state-of-the-art methods for learning model calibration and the traditional approach of measuring reliability diagrams. We empirically validate PL1 with experiments on eleven classification problems. Of particular note is that PL1 is shown to improve neural network calibration in almost all cases, which was shown to be unlikely with the aforementioned calibration approaches.

2 Calibration Methods

In this section we outline three current calibration methods for mapping model predictions to posterior probabilities, namely: binning, Platt scaling and isotonic regression, and discuss some of their practical aspects.

2.1 Methods of Model Calibration

Binning [7] is a non-parametric method that can be used for mapping when the shape of the mapping function is unknown. In binning, training patterns are sorted by model score into *n* bins of equal size. Then, given a test pattern *x*, it is assigned a bin according to its model output score, which is subsequently corrected to the measured probability that *x* belongs to class *c*, which is the fraction of training examples in the bin that belongs to *c* (see Figure 2). The number of bins is problem and model dependent, and must be chosen by cross-validation.

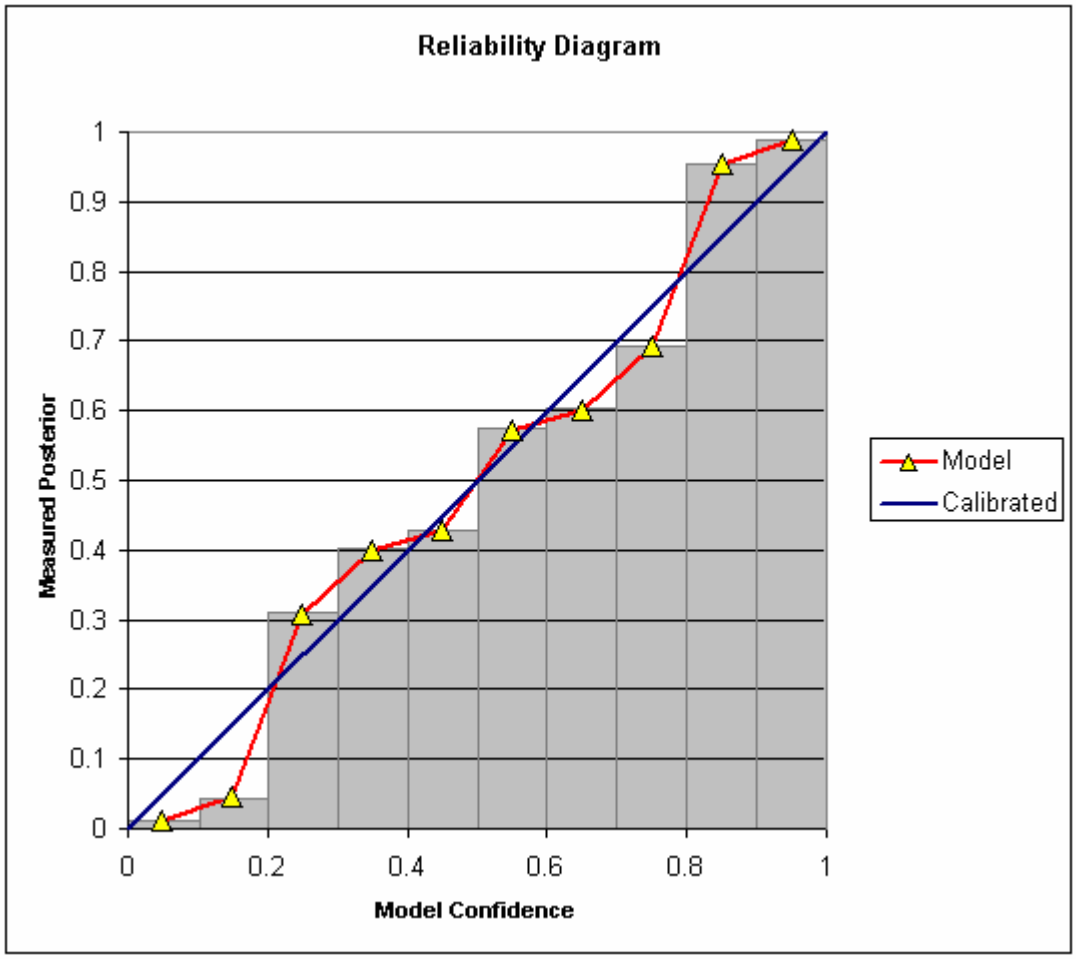

Figure 2. Calibration by binning.

Platt noticed a sigmoid-shaped distortion in comparing SVM outputs to predicted probabilities. His method attempts to correct this distortion by passing model outputs through a sigmoid [4]. The sigmoid's parameters are optimized using a maximum likelihood estimation from a fitting training set (see Figure 3). Fitting model outputs to a sigmoid has been shown to work well for some methods, but not for others, such as naïve Bayes [1,6].

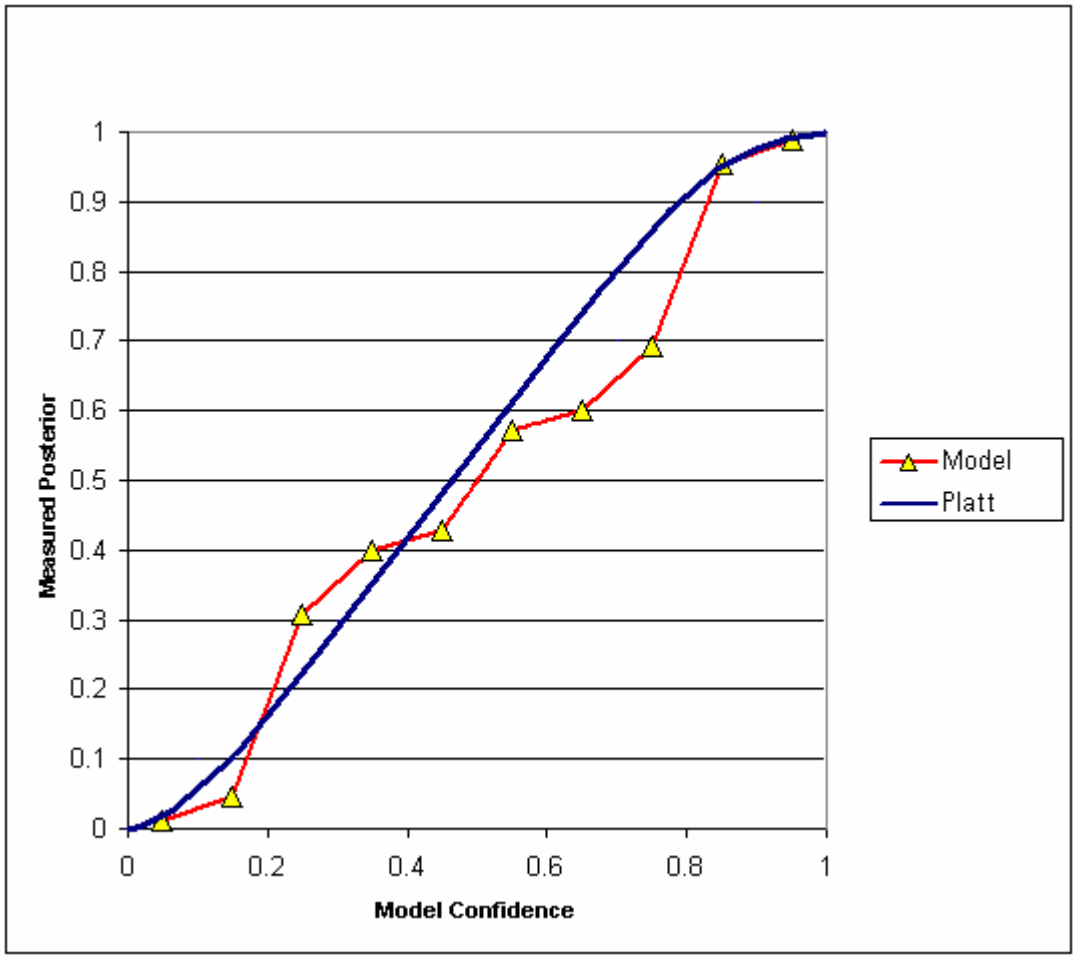

Figure 3. Sigmoid calibration curve generated by Platt's method.

A more general method for calibrating predictions based on *isotonic regression* was introduced by Zadrozny and Elkan in [5,6]. This method is a non-parametric form of regression that can be considered an intermediary approach between binning and sigmoid fitting, where the only restriction is requiring the calibration curve to be isotonic (monotonically increasing). A straightforward method for generating such a curve is the pair-adjacent violators (PAV) algorithm [8], which finds a stepwise constant solution (see Figure 4).

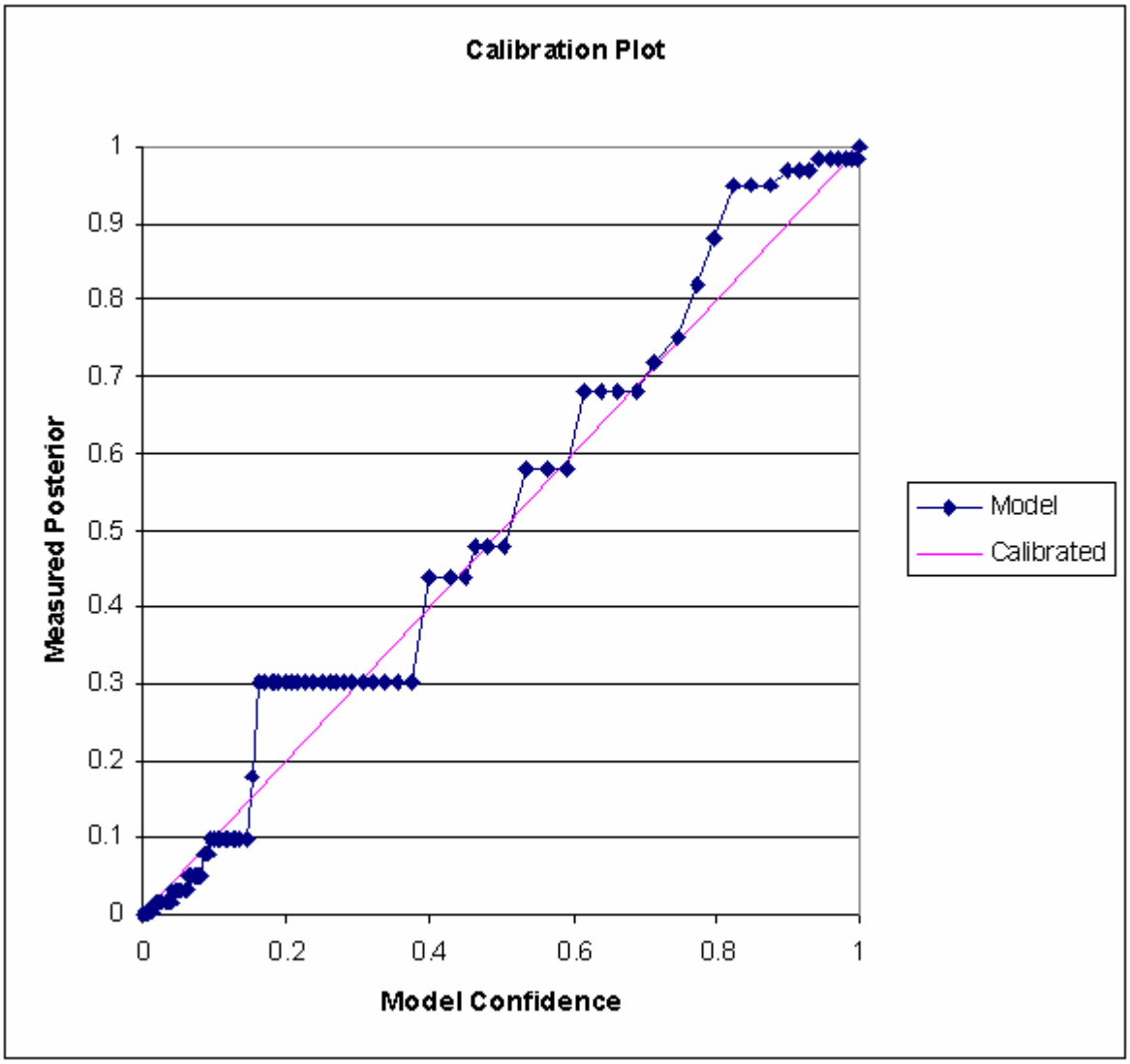

Figure 4. Isotonic regression curve generated using the PAV algorithm.

2.2 Practical considerations when calibrating

In order to avoid introducing unwanted bias, it is generally desirable that the data used for calibration be separate from the data used to train the model. In practice, this holdout set may be the same as the holdout set used to perform model parameter selection. In [1], Niculescu-Mizil and Caruana showed that Platt scaling generally performed better than isotonic regression using PAV when only a small amount of calibration data is available,

while isotonic regression usually outperformed Platt scaling with large amounts of calibration data.

Of these three methods, binning's approach of empirically predicting a probability for each bin and correcting model outputs to reflect that probability most directly matches the method of producing reliability diagrams and evaluating model calibration. However, binning has several undesirable artifacts. One is the necessity of choosing the number of bins by cross-validation. With small and unbalanced data sets, cross-validation is likely to indicate a suboptimal number of bins. Also, even when cross-validation yields a globally-optimal number of bins, this representation can underestimate or overestimate the optimal bin width for local regions of the curve. Furthermore, the size (domain) of bins is fixed and the boundaries are chosen arbitrarily. Binning gives the patterns of each bin a uniform posterior estimate and sharp jumps in probability between bins, which are probably incorrect. One conceivable way to mitigate this issue is by interpolating between bin centers to arrive at a more reliable probability [9].

In [1], empirical tests performed by Niculescu-Mizil and Caruana showed both Platt's method and isotonic regression to be unsuitable for neural network calibration. They explain that neural networks already produce well-calibrated probabilities when measured with conventional reliability diagrams, hence their reliability cannot be further improved. However, in Section 4, we show that neural network calibration may be improved by PL1.

3 PL1 algorithm

In this section we present PL1, a point-wise local binning method for measuring posterior probability, and describe how it may be used for model calibration and measuring reliability.

3.1 Algorithm

Without loss of generality, here we assume a two-class problem, with patterns labeled either positive or negative. PL1 takes a set of training patterns, T, of size N, sorted on the learning model's output, o, and measures a posterior probability for each pattern. For a given pattern, $i \in T$, with output o_i, a local window, W_i, of the $n \in T$ closest points with respect to o is considered to empirically estimate the posterior, $\hat{p}_i$, at model output o_i by tallying how many patterns in this window are of the positive class. Together, the set of points (o_i, $\hat{p}_i$) form the plot used for recalibrating model outputs on test patterns.

To calibrate a test pattern with output o, the two bounding points, L and R, on the calibration plot with respect to o are considered, and their $\hat{p}$ values interpolated to arrive at a recalibrated output value, o', as follows:

$$o' = \hat{p}_L + (\hat{p}_R - \hat{p}_L)\frac{o - o_L}{o_R - o_L} \quad (1)$$

For speed, in (1) we use linear interpolation, but cubic- or other forms of interpolation may be used by considering additional bounding points. Pseudo-code for PL1 is listed in Table 1.

Table 1. PL1 algorithm for estimating posterior probabilities from uncalibrated model predictions.

Sort training patterns, T, according to the learning model's output, o, on each pattern.
For each pattern, $i \in T$, with output o_i:
W_i = the set of n points closest to o_i, where
$n = \sqrt{\#\text{ patterns in the positive class}}$.
Estimate posterior $\hat{p}_i$ = fraction of patterns belonging to the positive class in W_i.
Add point $(o_i, \hat{p}_i)$ to the calibration plot.
Given a test pattern with output o, let the recalibrated model output o' = the interpolated p value of points bounding o on the calibration plot.

For example, if the learning model outputs a value of 0.7 on a test pattern, and the points bounding this value on the calibration plot are (0.65, 0.5) and (0.75, 0.7), then the recalibrated model output would be 0.5 + (0.7 – 0.5) * (0.7 – 0.65) / (0.75 – 0.65) = 0.6.

Although the calibration methods discussed in section 2 are designed for binary classification, multi-class problems may be considered by considering them as multiple binary problems, calibrating the binary models, and recombining the predictions [6].

This technique is likewise applicable to PL1, as follows:

For each class label *c*:

1. Generate a calibration plot, P_c, as outlined in Table 1, sorting on the learning model's output for *c*
2. Use P_c to calibrate the model output for *c* on test patterns.

That is, one calibration plot per class is generated after training and stored for use during the test phase. For multi-output models like neural networks, this procedure is simple and straightforward.

If too few training patterns are present within a problem class for probability measurements to be reliable (e.g. 30), then PL1 abstains from calibrating model outputs for that class.

3.2 Implementation details

After sorting the patterns by output value, a single linear scan through the data is all that is required to first determine the local window for the first point and then slide that window along the set of ordered points, re-centering it on each subsequent point. Hence, the time required to generate the calibration plot on sorted data is O(*N*).

We set window size *n* equal to the square root of the number of training patterns in the positive class to provide a balance between statistical reliability and precision. Assuming the training set is representative, as the number of patterns in the training set increases, so can the number of patterns included in each bin without losing representational accuracy.

There may be several points which are given the same value for *o* by the learning model. In this case, all patterns given an identical output are considered, allowing more than $\sqrt{\#\text{patterns}}$ in some local windows. That is, all patterns that are tied for the *n* closest patterns are included in the local window. Adding more patterns of the same distance increases statistical reliability of the empirical posterior measure $\hat{p}_i$ while not degrading precision.

Once the calibration plot is constructed, it will contain *N* (*x*,*y*) pairs, where *N* is the number of training points considered. If this number is unwieldy during deployment, it may be reduced by utilizing any number of binning or curve fitting approximation techniques. A discussion of these methods is outside the scope of this paper.

3.3 Comparison to existing methods

PL1 provides advantages over each of the above calibration methods. The main advantage is that, in contrast to the above methods, PL1 makes no assumptions about how the learning model treats the training data. The ramifications of this take many forms, described below.

Unlike binning, PL1 does not have an arbitrary partitioning of examples into fixed-width bins or need to determine the number of bins by cross-validation. A globally-optimal bin width or number of bins is not assumed. Instead, the bin width is fit to each training point based on local context. PL1 does not assume all patterns in each bin have a single common posterior, so bins are allowed to partially overlap.

PL1 improves on Platt's method because it does not assume a specific shape distortion of pattern output values (e.g., sigmoidal), but is able to fit any shape, given enough data to match that shape's complexity. However, PL1 does not require a large amount of data in order to take effective measurements for posterior estimation since point bins may overlap.

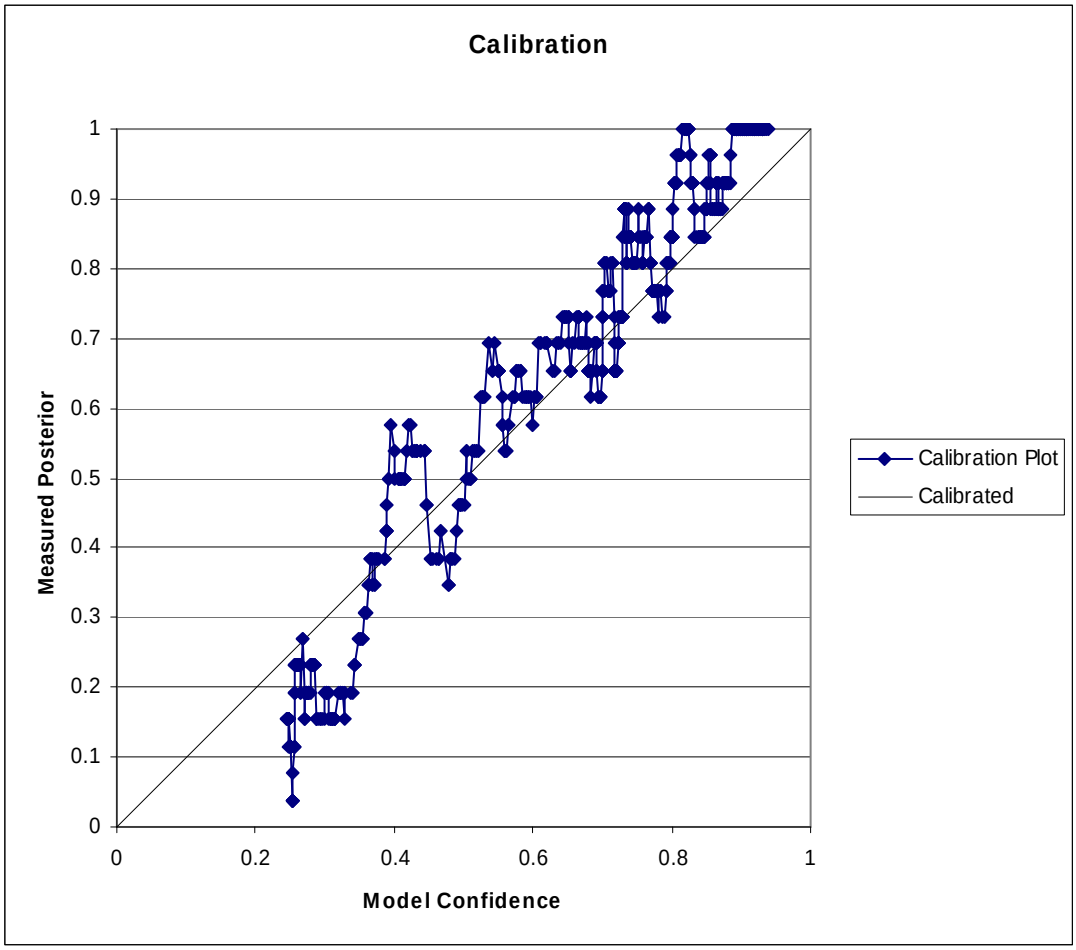

Figure 5. Sample calibration plot generated by PL1.

Figure 5 shows a representative calibration plot generated by PL1. Observe that curves generated by PL1 are not isotonic. Constraining the calibration curve to be isotonic is

valid when the assumption that the learning model ranks examples correctly is met. However, this is not true for every model. For instance, consider a linear model (e.g., a perceptron) attempting to learn a problem that is not linearly separable.

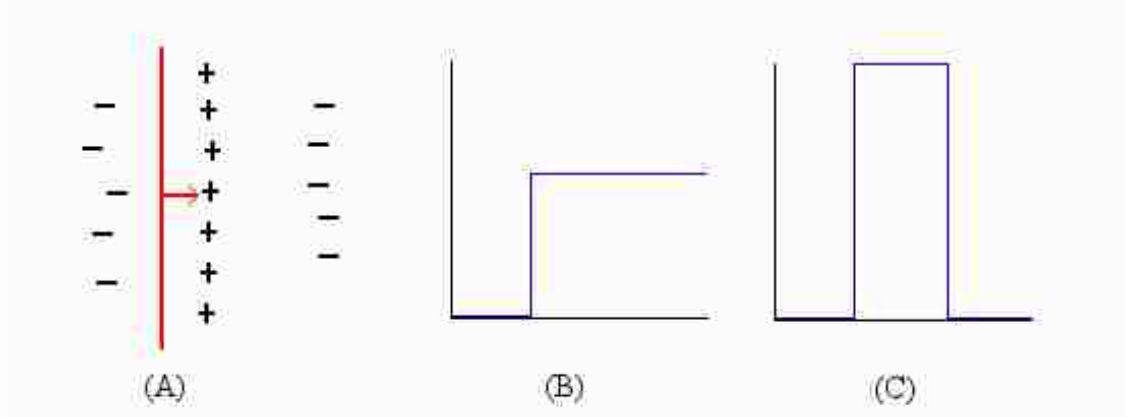

Figure 6. A simple non-linearly separable problem (A), a calibration curve for a perceptron attempting to learn this problem constructed via isotonic regression (B), and an optimal calibration plot (C).

For the hypothetical data set in Figure 6, the non-linear nature of the problem precludes the use of an isotonic curve to perform optimal calibration for a linear learning model. Here, the problem is too complex for a perceptron to learn (Figure 6a) and an isotonic curve does not allow for optimal model calibration (Figure 6b). If the restriction of a non-decreasing probability output ranking is relaxed, then an optimal calibration plot may be attained (Figure 6c). Therefore, we consider it an improvement to remove the assumption that examples are ranked correctly by the learning model, without hindering the ability to generate an isotonic curve when this assumption is met.

Observe that isotonic calibration curves do not change the ranking of patterns, but non-isotonic calibration can alter pattern ranking. PL1 has greater power to alter a model's

outputs, and hence more potential to alter classification accuracy. In the case presented in Figure 6, this property could improve the model's classification accuracy toward the optimal value. For other problems, it is also possible the model's overall classification accuracy could be reduced. However, considering that the purpose for calibrating model outputs is to perform in problem domains where good probability values are important as well as classification accuracy, it is acceptable to tradeoff some classification accuracy for improved probability outputs.

As an aside, calibration algorithms that are better able to fit a model's actual performance on a data set not only improve model reliability, but could also be used to study the general suitability of various families and types of learning models to the task of learning various problem domains. For example, a posterior curve such as the one shown in Figure 6c might show that the learning model is underpowered to correctly learn the problem at hand and a more powerful model ought to be used. Such a study, however, is outside the scope of this work.

If obtaining an isotonic function is desired, the PAV algorithm may be applied to the calibration plots generated by PL1 as a post-processing step.

3.4 PL1 applied to plotting reliability diagrams

The problems with using binning for generating calibration curves discussed in section 3.3 also exist with the traditional method for plotting reliability diagrams (i.e. using a set number of exclusive bins of equal width across the output range can lead to inaccurate

results). In [1], Niculescu-Mizil and Caruana found that neural networks appear fairly well-calibrated on reliability diagrams (see Figure 3). However, PL1 shows models that are less well-calibrated in comparison.

The model shown in Figure 2 above exhibits a mean-squared error (MSE) of 0.0030 using the conventional 10-bin method, approximately a 5% mean error in posterior estimation. In contrast, using PL1 to plot reliability calculates a MSE of 0.0103, approximately a 10% mean error in posterior estimation (see Figure 7). The 10-bin method overestimates the model's reliability due to the issues discussed in section 2.2.

Figure 7 shows reliability plots generated using our method before and after point-wise calibration for the learning model used in Figure 2. MSE before and after calibration was 0.0103 (10% mean error) and 0.0059 (7.5% mean error), respectively. However, on other data sets or with other learning models, conventional reliability diagrams may underestimate the model's calibration compared to PL1.

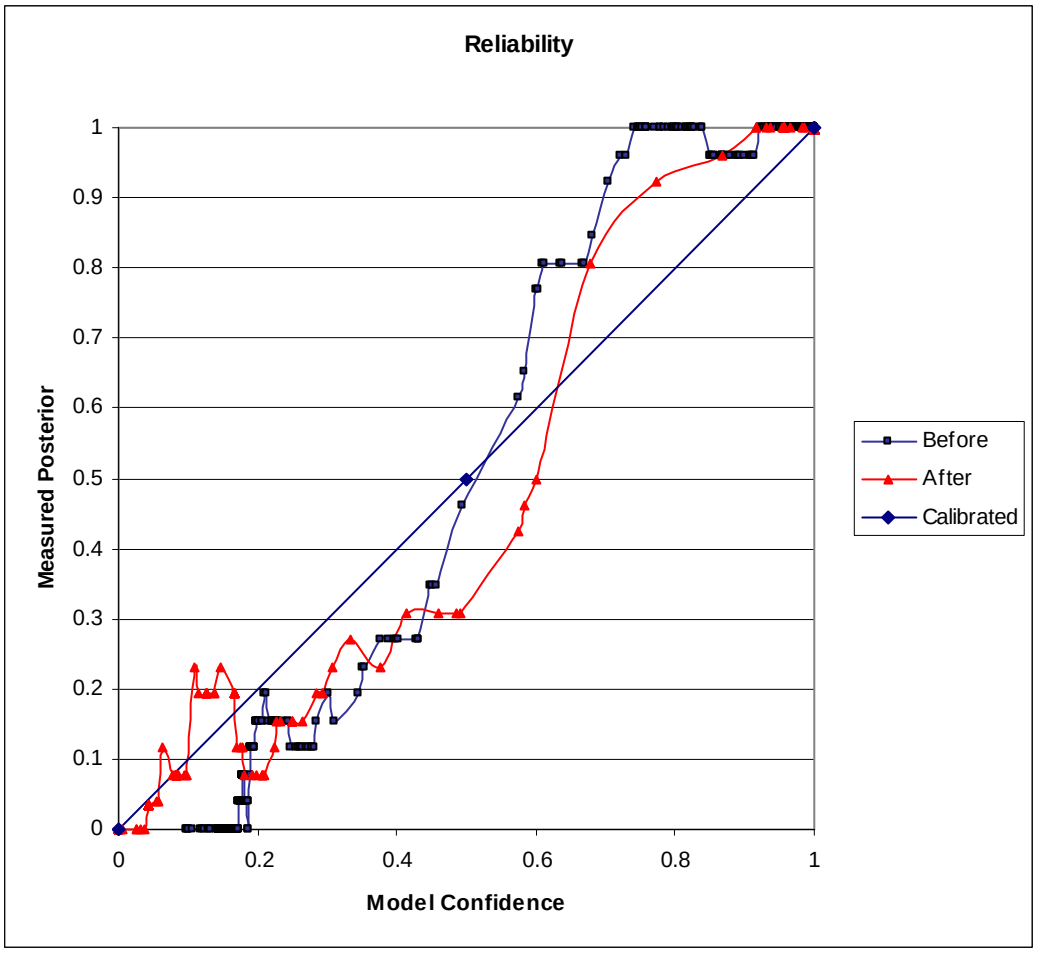

Figure 7. Reliability diagram constructed using PL1 to measure reliability before and after calibration by PL1.

4 Experiments

Eleven well-known benchmark classification problems were selected from the UC Irvine Machine Learning Repository (UCI MLR) [10]. The problems were selected so as to have a variety of characteristics (number of patterns, number and type of features, number of class labels, and complexity) in order to analyze the robustness of our calibration method (see Table 2).

Table 2. Properties of benchmark data sets.

Data set	Patterns	Features	Classes
ann	7200	21	3
balance	625	4	3
bcw	699	9	2
derm	358	34	6
ecoli	336	7	8
ionosphere	351	33	2
iris	150	4	3
musk2	6598	166	2
pima	768	8	2
sonar	208	60	2
wine	178	13	3

Experiments were performed on four learning algorithms: artificial neural networks (ANN), naïve Bayes, a bagged ensemble of 100 ID3 decision trees (100-BDT), and *k*-NN with $k=\sqrt{\#\,\text{patterns}}$.

Feature values (both nominal and continuous) were normalized between zero and one. Pattern classification was determined by *winner-take-all* (the class of the output node with the highest value is chosen). Testing was performed using 10-fold stratified cross-validation. Each experiment was rerun ten times and the results averaged.

The ANN models were standard feed-forward multi-layer perceptron networks optimizing cross-entropy, both with and without weight decay regularization. The multi-layer perceptrons had a single, fully connected hidden layer and were trained through on-line backpropagation. The optimal number of hidden nodes was empirically determined for each task with a validation set, searching layer sizes within the range from one to fifty hidden nodes. Networks had one output node per class (including on two-class

problems). In all experiments, network weights were initialized to uniform random values within the range [-0.01,0.01] [11]. Training patterns were presented to the network in a random order each epoch. Learning rate and momentum were optimized for each application by cross-validation. Weight decay values of λ = 0, 0.00001, 0.00003, 0.0001, 0.0003, and 0.001 were considered and the optimal value used for each application [12]. Training continued until the training set was successfully learned or training set classification error ceased to decrease for a substantial number of epochs. The model selected for test evaluation was the network on the epoch with the best holdout set accuracy, where the holdout set consisted of 20% of the original training data.

Results are displayed in Tables 3-6. Each row displays the averaged results on one data set. The left half of each table (columns 2-4) displays classification accuracy before calibration and after calibration with PL1 and isotonic regression with PAV. The right half (columns 5-7) displays MSE before and after PL1 and PAV calibration. We chose to display MSE because it is a proper scoring rule [3] that is minimized as calibration is improved.

Table 3. ANN performance on benchmark data sets.

Dataset	Classification Accuracy	Accuracy after PL1	Accuracy after PAV	MSE	MSE after PL1	MSE after PAV
Ann	0.9909	0.9909	0.9909	0.0106	0.0050	0.0050
Balance	0.9344	0.9320	0.9296	0.0488	0.0364	0.0366
Bcw	0.9717	0.9718	0.9715	0.0216	0.0224	0.0225
Derm	0.9736	0.9745	0.9761	0.0144	0.0081	0.0072
Ecoli	0.8297	0.8218	0.8187	0.0805	0.0340	0.0598
Ionosphere	0.9200	0.9280	0.9277	0.0624	0.0593	0.0598
Iris	0.9580	0.9533	0.9520	0.0321	0.0192	0.0203
Musk2	0.9869	0.9864	0.9869	0.0162	0.0123	0.0123
Pima	0.7661	0.7645	0.7608	0.1565	0.1578	0.1579
Sonar	0.8426	0.8429	0.8400	0.1252	0.1241	0.1248
Wine	0.9822	0.9789	0.9789	0.0195	0.0134	0.0128
Average	**0.9233**	**0.9223**	**0.9212**	**0.0534**	**0.0447**	**0.0472**

Though not the primary goal of model calibration, it is of interest to note model test accuracy before and after calibration (Table 3, columns 2-4). Classification accuracy is approximately the same before and after calibration. With PAV, accuracy decreases by 0.2%, and with PL1, average accuracy decreases by 0.1%.

Columns 5-7 show MSE without calibration, after PL1, and after PAV, respectively. For ANNs (Table 3), improvement is shown in almost all cases, except for *bcw* and *pima*. MSE is reduced 15%, from 0.0589 to 0.0502 with PL1, and reduced 10.5% to 0.0526 with PAV.

Table 4. 100-BDT performance on benchmark data sets.

Dataset	Classification Accuracy	Accuracy after PL1	Accuracy after PAV	MSE	MSE after PL1	MSE after PAV
Ann	0.9990	0.9992	0.9992	0.0005	0.0005	0.0005
Balance	0.9760	0.9760	0.9760	0.0189	0.0147	0.0147
Bcw	0.9971	0.9941	0.9941	0.0057	0.0033	0.0041
Derm	0.9888	0.9888	0.9888	0.0037	0.0033	0.0033
Ecoli	0.9881	0.9851	0.9851	0.0067	0.0034	0.0032
Ionosphere	0.9972	0.9972	0.9972	0.0109	0.0030	0.0030
Iris	1.0000	1.0000	1.0000	0.0038	0.0007	0.0004
Musk2	0.9985	0.9986	0.9986	0.0033	0.0010	0.0010
Pima	0.9714	0.9714	0.9701	0.0390	0.0248	0.0258
Sonar	0.9712	0.9712	0.9712	0.0293	0.0159	0.0159
Wine	1.0000	1.0000	1.0000	0.0045	0.0006	0.0006
Average	**0.9897**	**0.9892**	**0.9891**	**0.0115**	**0.0065**	**0.0066**

Accuracy is practically unchanged when calibrating the decision tree ensemble. MSE is reduced by more than 40% after calibration. PL1 and PAV exhibit roughly the same performance on average.

Table 5. Naïve Bayes performance on benchmark data sets.

Dataset	Classification Accuracy	Accuracy after PL1	Accuracy after PAV	MSE	MSE after PL1	MSE after PAV
Ann	0.9549	0.9608	0.9613	0.0242	0.0217	0.0213
Balance	0.9152	0.9136	0.9168	0.0797	0.0530	0.0491
Bcw	0.9678	0.9649	0.9649	0.0315	0.0330	0.0323
Derm	0.9022	0.9078	0.9106	0.0183	0.0176	0.0175
Ecoli	0.6577	0.7530	0.8006	0.0762	0.0480	0.0363
Ionosphere	0.8519	0.8946	0.8917	0.1399	0.0917	0.0867
Iris	0.9533	0.9600	0.9533	0.0242	0.0265	0.0279
Musk2	0.8459	0.8459	0.8459	0.2500	0.2500	0.2500
Pima	0.7513	0.7487	0.7500	0.1786	0.1751	0.1691
Sonar	0.6731	0.6923	0.7260	0.2943	0.2041	0.1942
Wine	0.9719	0.9775	0.9719	0.0140	0.0134	0.0164
Average	**0.8587**	**0.8745**	**0.8812**	**0.1028**	**0.0849**	**0.0819**

Table 6. *k*-NN performance on benchmark data sets.

Dataset	Classification Accuracy	Accuracy after PL1	Accuracy after PAV	MSE	MSE after PL1	MSE after PAV
Ann	0.9269	0.9360	0.9363	0.0408	0.0373	0.0371
Balance	0.8416	0.8336	0.8352	0.1108	0.0830	0.0843
Bcw	0.9414	0.9649	0.9678	0.0393	0.0289	0.0281
Derm	0.6620	0.6927	0.6816	0.0892	0.0753	0.0764
Ecoli	0.8482	0.8452	0.8482	0.0298	0.0288	0.0280
Ionosphere	0.8348	0.8832	0.8832	0.1379	0.0907	0.0902
Iris	0.9800	0.9800	0.9800	0.0177	0.0167	0.0178
Musk2	0.9323	0.9315	0.9303	0.0517	0.0460	0.0459
Pima	0.7396	0.7331	0.7383	0.1741	0.1774	0.1745
Sonar	0.6683	0.6635	0.6346	0.2019	0.2044	0.2005
Wine	0.6966	0.6966	0.6966	0.1170	0.1213	0.1180
Average	**0.8247**	**0.8328**	**0.8302**	**0.0918**	**0.0827**	**0.0819**

PL1 and PAV calibration significantly improve classification accuracy and MSE for naïve Bayes and k-NN. PAV shows better classification improvement for naïve Bayes than PL1, and PL1 performs better than PAV on k-NN. PL1 and PAV improve MSE on naïve Bayes and k-NN, with PAV performing slightly better on average.

5 Conclusions

In this work PL1, a new algorithm for performing empirical estimates of model posteriors, was presented. This technique may be used in constructing model calibration curves for outputting more accurate posterior probabilities. PL1 outperforms isotonic regression using the pairwise-adjacent violators algorithm (PAV) when calibrating neural networks. Where prior work [1] has shown PAV [4,5,6] unable to improve neural network reliability, applying PAV on these data sets does show an improvement in error.

PL1 generally outperforms PAV and exhibits a significant reduction in neural network calibration error on nearly all data sets tested. PL1 was shown to perform well for data sets and learning models of varying size and complexity, reducing MSE for neural network models by 15% on average. PL1 performs comparably to PAV when calibrating a bagged decision tree ensemble. On naïve Bayes, PAV performs better than PL1, and with k-NN, PL1 and PAV perform comparably.

We plan to apply PL1 to other learning algorithms, such as classification-based error functions for neural networks [13], which have been shown to yield higher classification accuracy than cross-entropy optimization, but do not output accurate posteriors. It is expected that post-training calibration with PL1 will improve these networks' posterior probability prediction while allowing them to retain their higher level of classification accuracy.

References

1. Niculescu-Mizil, A. & Caruana, R. (2005). Predicting Good Probabilities with Supervised Learning. Proceedings of the American Meteorology Conference (AMS2005), San Diego.

2. DeGroot, M. & Fienberg, S. (1982). The comparison and evaluation of forecasters. *Statistician*, 32, pp. 12-22.

3. Brier, G. (1950). Verification of forecasts expressed in terms of probability. *Monthly Weather Review*, 75, pp. 1-3.

4. Platt, J. (1999). Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. *Advances in Large Margin Classifiers*, pp. 61-74.

5. Zadrozny, B. & Elkan, C. (2001). Obtaining calibrated probability estimates from decision trees and naïve Bayesian classifiers. *International Conference on Machine Learning*, pp. 609-616.

6. Zadrozny, B. & Elkan, C. (2002). Transforming classifier scores into accurate multiclass probability estimates. *Knowledge Discovery and Data Mining*, pp. 694-699.

7. Zadrozny, B. & Elkan, C. (2001). Learning and making decisions when costs and probabilities are both unknown. In *Proceedings of the Seventh International Conference on Knowledge Discovery and Data Mining*, pp. 204-213. ACM Press.

8. Ayer, M., Brunk, H., Ewing, G., Reid, W., & Silverman, E. (1955). An empirical distribution function for sampling with incomplete information. *Annals of Mathematical Statistics*, 5, pp. 641–647.

9. Wilson, D. & Martinez, T. (1997). Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research* (JAIR), 1, pp. 1–34.

10. Blake, C. and Merz, C. (1998). UCI machine learning repository. http://www.ics.uci.edu/~mlearn/MLRepository.html. Irvine, CA: University of California, Department of Information and Computer Science.

11. Thimm, G. and Fiesler, E. (1997). High-order and multilayer perceptron initialization. IEEE Transactions on *Neural Networks*, 8:2, pp. 249-259.

12. Krogh, A. & Hertz, J. (1992). A simple weight decay can improve generalization. *Advances in Neural Information Processing Systems (NIPS)* 4, pp. 950-957, San Mateo, CA.

13. Rimer, M. & Martinez, T. (2006). Classification-based Objective Functions. *Machine Learning* 63:2, pp. 183-205.

Chapter 8

Calibrating Classification-based Networks to Improve Posteriors

Michael Rimer and Tony Martinez
Department of Computer Science, Brigham Young University, Provo, UT 84602

Abstract. Classification-based (CB) error functions can improve neural network classification accuracy over conventional objective functions like cross-entropy (CE). However, resultant model outputs are not usable as posterior probabilities, as opposed to CE trained networks. This work applies a new point-wise local binning method, PL1, for calibrating CB model outputs to provide more accurate probability predictions. Empirical tests show that calibrated CB networks can yield more accurate posteriors than uncalibrated CE networks while retaining a significantly higher degree of accuracy. When calibrated, CB and CE models yield equally accurate posterior probabilities, while CB models remain more accurate.

1 Introduction

In many machine learning applications, it is important for the learning model to yield accurate posteriors. This is applicable to problem domains with unknown or asymmetric cost functions or in multi-tier learning systems, where it is desired that the outputs of one layer of the system be used as probabilities in helping the next layer make a decision. In these situations, achieving high classification accuracy may not be sufficient. Certain learning methods output values that may be used naturally as posteriors, such as softmaxed cross-entropy (CE) neural networks and bagged decision tree ensembles.

Other learning models do not output posteriors, such as decision trees, or may output biased posteriors (e.g. naïve Bayes and SVMs) [1].

Classification-based (CB) error functions [2,3,4] are learning algorithms that implement objective functions designed to maximize classification accuracy. Though more accurate with respect to classification accuracy than CE networks, CB networks are not designed to yield output values usable as accurate posterior probabilities.

Reliability diagrams [5] are commonly used in visualizing model calibration for real problem domains where true conditional probabilities are not known. The accepted method for generating these diagrams involves discretizing the prediction space into a set number of exclusive bins and plotting the mean predicted value against the true (i.e., empirically measured) fraction of positive cases. For a well-calibrated model, predictions will fall along the diagonal line, representing a model whose confidence matches the actual posterior probability for a given data set (see Figure 1). Calibration mean-squared error (CMSE) is calculated by averaging the squared distance of the model's prediction from the measured posterior across all bins.

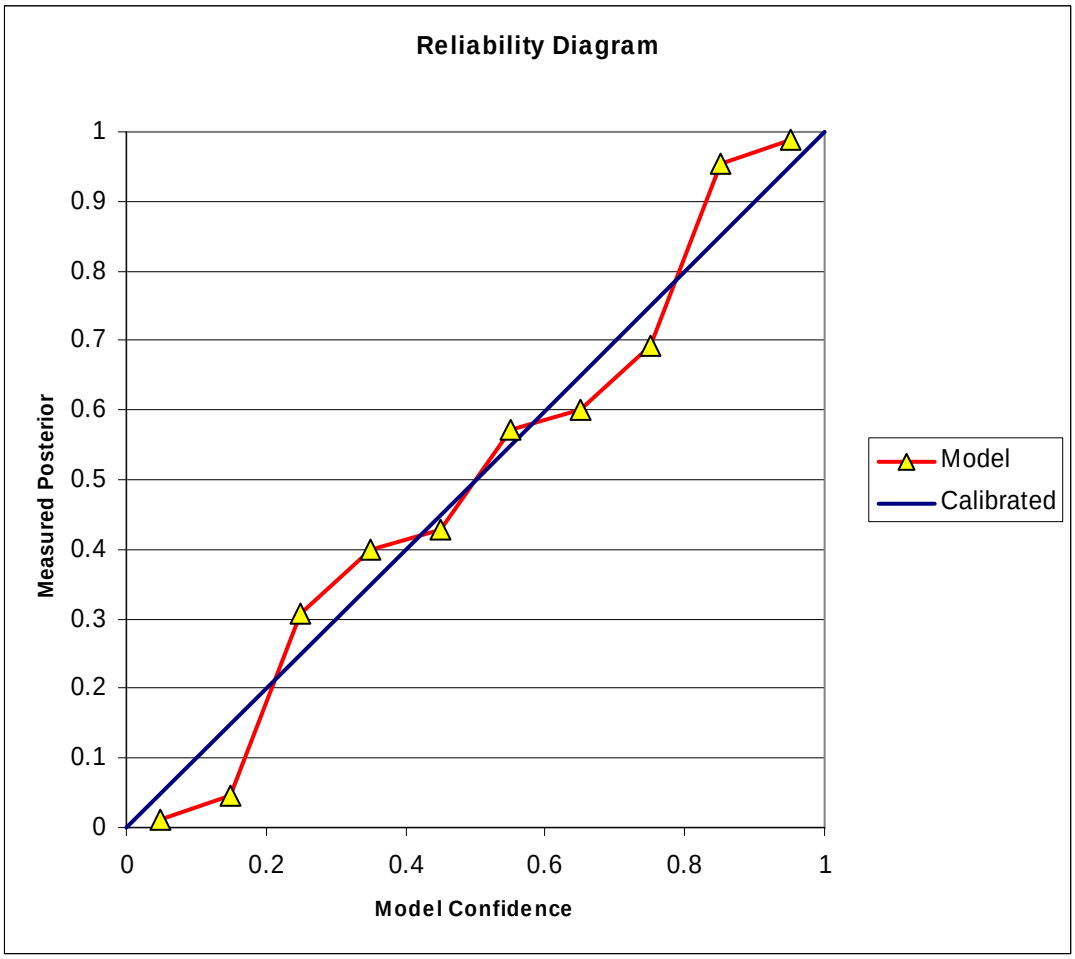

Figure 1. Reliability diagram using the conventional 10-bin method.

In certain cases, introducing a post-processing step of model output calibration, such as Platt Scaling [6] or Isotonic Regression [7,8] may yield more accurate or less biased posteriors. Niculescu-Mizil and Caruana [1] showed that using reliability diagrams, these methods may be effective in improving model calibration for boosted trees, random forests, and SVMs, while not reducing or increasing calibration error for neural networks. In [9], a new method for model calibration and reliability diagram generation, PL1, was proposed, which was shown to improve neural network calibration.

In this work, we use PL1 to calibrate CB trained neural networks with the aim of making their outputs reflect posterior probabilities. We empirically validate this method with experiments on eleven classification problems and show that PL1 is able to significantly reduce calibration error of CB networks, making their use feasible in problem domains where outputting accurate posterior probabilities is important. Calibrated CB networks are shown to have calibration MSE equal to CE trained networks, while also retaining their higher degree of accuracy.

2 Calibration Methods

In this section we provide a brief overview of three calibration methods for mapping model predictions to posterior probabilities: binning, Platt scaling and isotonic regression, and discuss their applicability to calibrating neural networks.

2.1 Methods

Binning [10] is a non-parametric method that can be used for mapping when the shape of the mapping function is unknown. In binning, training patterns are sorted by model score into n bins of equal size. Then, given a test pattern x, it is assigned a bin according to its model output score, which is subsequently corrected to the measured probability that x belongs to class c, which is the fraction of training examples in the bin that belongs to c (see Figure 2). The number of bins is problem and model dependent, and must be chosen by cross-validation.

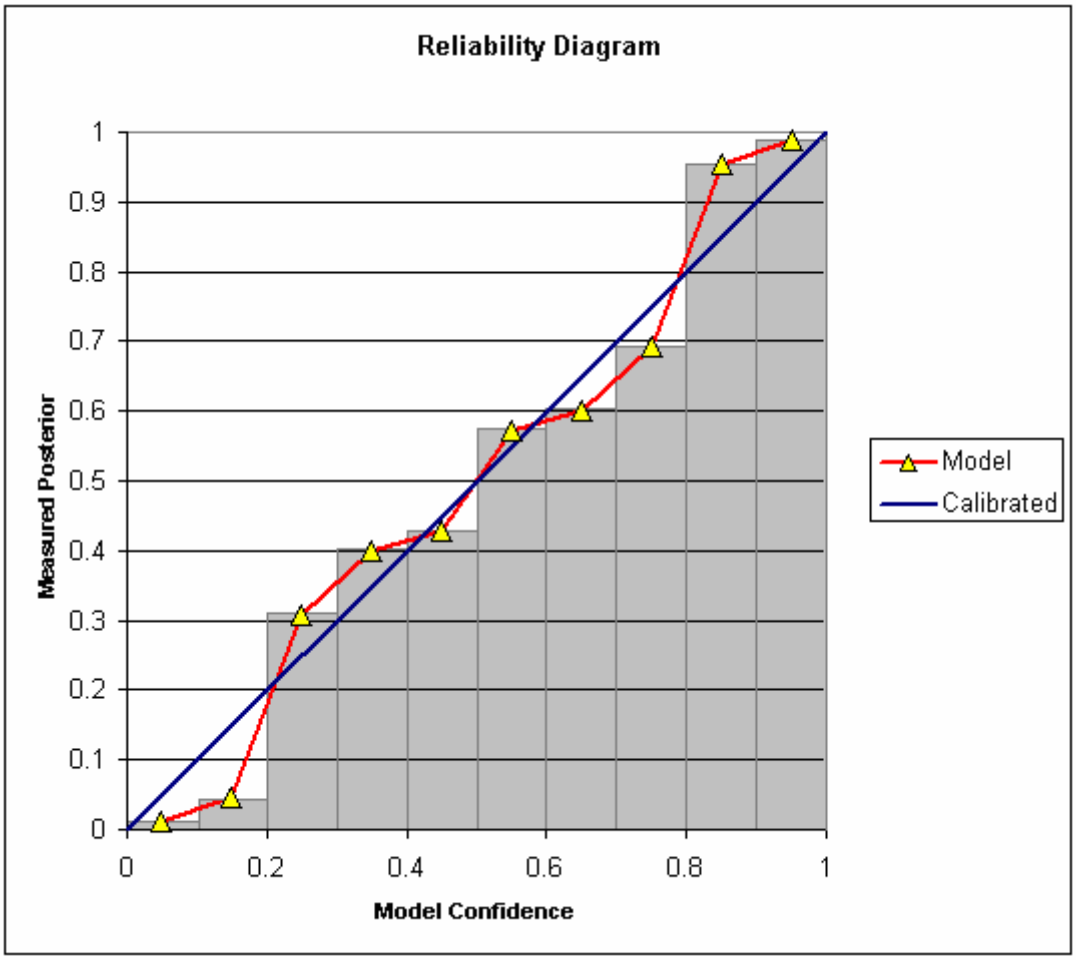

Figure 2. Calibration by binning.

Platt noticed a sigmoid-shaped distortion in comparing SVM outputs to predicted probabilities. His method attempts to correct this distortion by passing model outputs through a sigmoid [5]. The sigmoid's parameters are optimized using a maximum likelihood estimation from a fitting training set (see Figure 3). Fitting model outputs to a sigmoid has been shown to work well for some methods, but not for others, such as naïve Bayes [1,7].

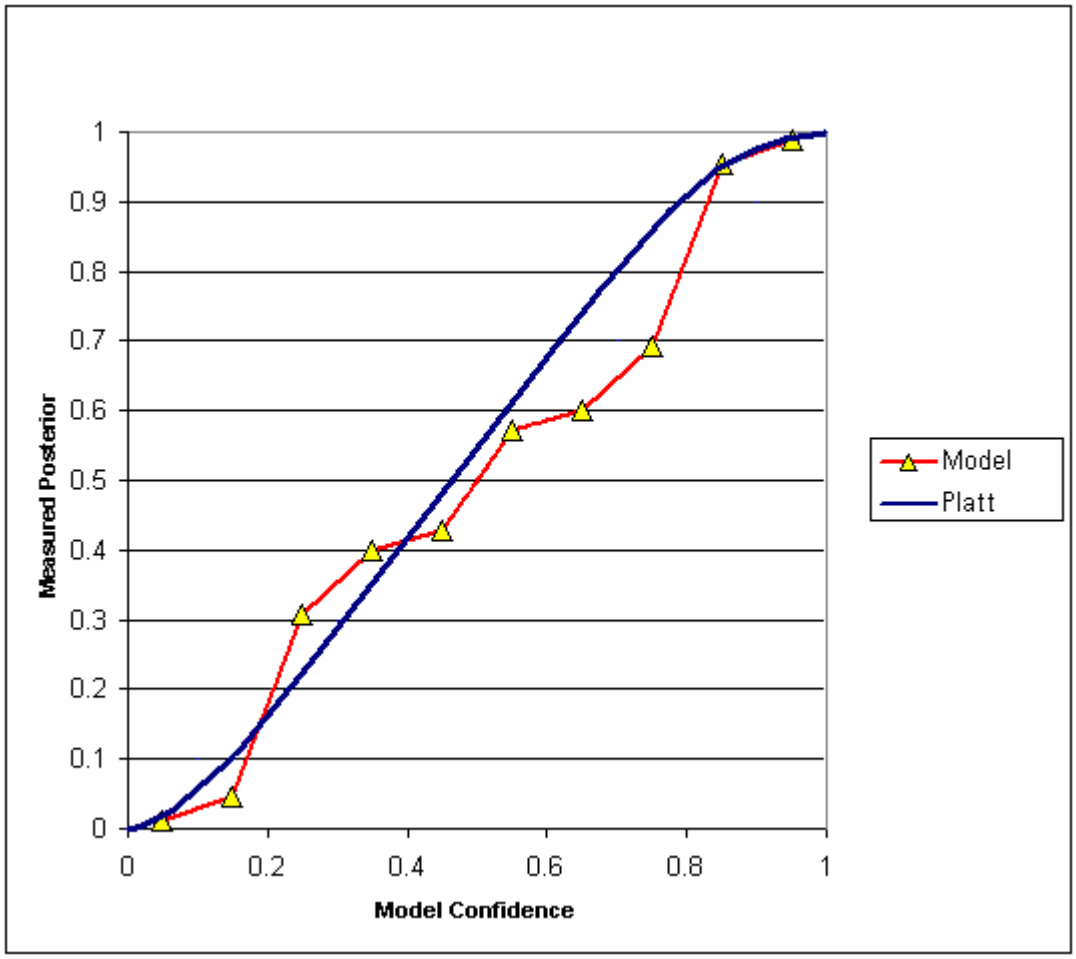

Figure 3. Sigmoid calibration curve generated by Platt's method.

A more general method for calibrating predictions based on *isotonic regression* was introduced by Zadrozny and Elkan in [6,7]. This method is a non-parametric form of regression that can be considered an intermediary approach between binning and sigmoid fitting, where the only restriction is requiring the calibration curve to be isotonic (monotonically increasing). A straightforward method for generating such a curve is the pair-adjacent violators (PAV) algorithm [11], which finds a stepwise constant solution (see Figure 4).

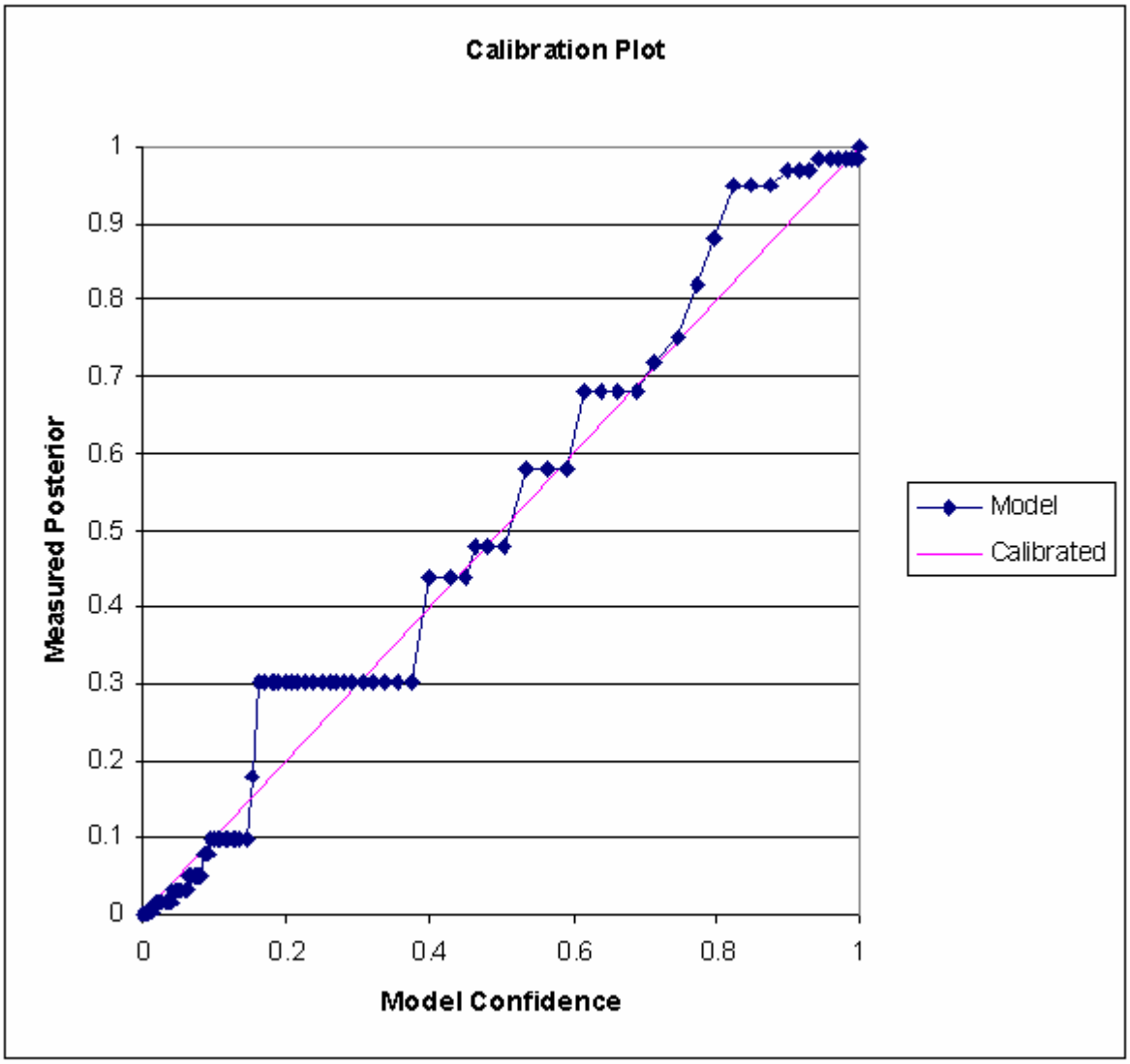

Figure 4. Isotonic regression curve generated using the PAV algorithm.

2.2 Applicability of above methods to neural network calibration

Binning has several undesirable artifacts. One is the necessity of choosing the number of bins by cross-validation. With small and unbalanced data sets, cross-validation is likely to indicate a suboptimal number of bins. Also, even when cross-validation yields a globally-optimal number of bins, this representation can underestimate or overestimate the optimal bin width for local regions of the curve. Furthermore, the size (domain) of bins is fixed and the boundaries are chosen arbitrarily. Binning gives the patterns of each

bin a uniform posterior estimate, which is probably an incorrect assumption. Similarly, from an instance-based perspective, for test patterns lying close to bin boundaries, it does not follow that a point on the opposite end of the bin should be considered relevant, while patterns from an adjacent bin that are physically closer are discounted. One conceivable way to mitigate this issue is by interpolating between bin centers to arrive at a more reliable probability [12].

In order to avoid introducing unwanted bias, it is generally desired for the data used for calibration to be separate from the data used to train the model. In practice, this holdout set may be the same as the holdout set used to perform model parameter selection. In [1], Niculescu-Mizil and Caruana showed that Platt scaling generally performed better than isotonic regression using PAV when only a small amount of calibration data is available, while isotonic regression usually outperformed Platt scaling with large amounts of calibration data.

In [1], empirical tests performed by Niculescu-Mizil and Caruana showed both Platt's method and isotonic regression to be unsuitable for neural network calibration. They explain that neural networks already produce well-calibrated probabilities when measured with conventional reliability diagrams, hence their reliability cannot be further improved. However, in [9] it was shown that using PL1 for reliability measurement and calibration, it is possible to further improve neural network posteriors. Therefore, we choose this method for calibrating CB trained networks in this work.

3 Point-wise Local Binning Calibration Method

In this section we briefly present the PL1 calibration method introduced in [9].

Without loss of generality, here we assume a two-class problem, with patterns labeled either positive or negative. PL1 takes a set of training patterns, T, of size N, sorted on the learning model's output, o, and measures a posterior probability for each pattern. For a given pattern, $i \in T$, with output o_i, a local window, W_i, of the $n \in T$ closest points with respect to o is considered to empirically estimate the posterior, $\hat{p}_i$, at model output o_i by tallying how many patterns in this window are of the positive class. Together, the set of points (o_i, $\hat{p}_i$) form the plot used for recalibrating model outputs on test patterns.

To calibrate a test pattern with output o, the two bounding points, L and R, on the calibration plot with respect to o are considered, and their $\hat{p}$ values interpolated to arrive at a recalibrated output value, o', as follows:

$$o' = \hat{p}_L + (\hat{p}_R - \hat{p}_L)\frac{o - o_L}{o_R - o_L} \tag{1}$$

For speed, in (1) we use linear interpolation, but cubic- or other forms of interpolation may be used by considering additional bounding points. Pseudo-code for PL1 is listed in Table 1.

Table 1. PL1 algorithm for estimating posterior probabilities from uncalibrated model predictions.

Sort training patterns, T, according to the learning model's output, o, on each pattern.
For each pattern, $i \in T$, with output o_i:
W_i = the set of n points closest to o_i, where
$n = \sqrt{\#\text{ patterns in the positive class}}$.
Estimate posterior $\hat{p}_i$ = fraction of patterns belonging to the positive class in W_i.
Add point $(o_i, \hat{p}_i)$ to the calibration plot.
Given a test pattern with output o, let the recalibrated model output o' = the interpolated p value of points bounding o on the calibration plot.

For example, if the learning model outputs a value of 0.7 on a test pattern, and the points bounding this value on the calibration plot are (0.65, 0.5) and (0.75, 0.7), then the recalibrated model output would be 0.5 + (0.7 – 0.5) * (0.7 – 0.65) / (0.75 – 0.65) = 0.6.

Although the calibration methods discussed in section 2 are designed for binary classification, multi-class problems may be considered by considering them as multiple binary problems, calibrating the binary models, and recombining the predictions [7].

The main advantage of this approach over the methods outlined in section 2 is that it makes no assumptions about how the model treats the training data. Ramifications of this are described in [9].

4 Experiments

Eleven well-known benchmark classification problems were selected from the UC Irvine Machine Learning Repository (UCI MLR) [13]. The problems were selected so as to

have a variety of characteristics (number of patterns, number and type of features, number of class labels, and complexity) in order to analyze the robustness of applying PL1 to CB trained networks method (see Table 2).

Table 2. Properties of benchmark data sets.

Data set	Patterns	Features	Classes
ann	7200	21	3
balance	625	4	3
bcw	699	9	2
derm	358	34	6
ecoli	336	7	8
ionosphere	351	33	2
iris	150	4	3
musk2	6598	166	2
pima	768	8	2
sonar	208	60	2
wine	178	13	3

Experiments were performed comparing optimized CE feed-forward multi-layer perceptron networks, both with and without weight decay regularization, to existing CB algorithms, CB1 [2], CB2 [3], and CB3 [4]. All networks had a single, fully connected hidden layer and were trained through on-line backpropagation. The optimal number of hidden nodes was empirically determined for each task with a validation set, searching layer sizes within the range from one to fifty hidden nodes. Networks had one output node per class (including on two-class problems). In all experiments, network weights were initialized to uniform random values within the range [-0.01,0.01] [14]. The learning rate and momentum were optimized for each task by cross-validation. For CE networks, weight decay values of λ = 0, 0.00001, 0.00003, 0.0001, 0.0003, and 0.001 were considered and the optimal value used for each application [15].

Feature values (both nominal and continuous) were normalized between zero and one. Training patterns were presented to the network in a random order each epoch. Each experiment was repeated ten times with a different random seed and the results averaged.

Pattern classification was determined by *winner-take-all* (the class of the output node with the highest value is chosen). Training continued until the training set was successfully learned or training set classification error ceased to decrease for a substantial number of epochs. The model selected for test evaluation was the network on the epoch with the best holdout set accuracy, where the holdout set consisted of 20% of the original training data.

Figure 2 displays 95% confidence intervals for the area under the ROC curve (AUC-ROC) for each learning method, averaged across all experiments. The ROC curve is used to measure a model's performance when using different threshold values as the cutoff point for labeling patterns as positive. AUC-ROC measures the model's robustness to using cutoff values across the entire range of possible cutoff points.

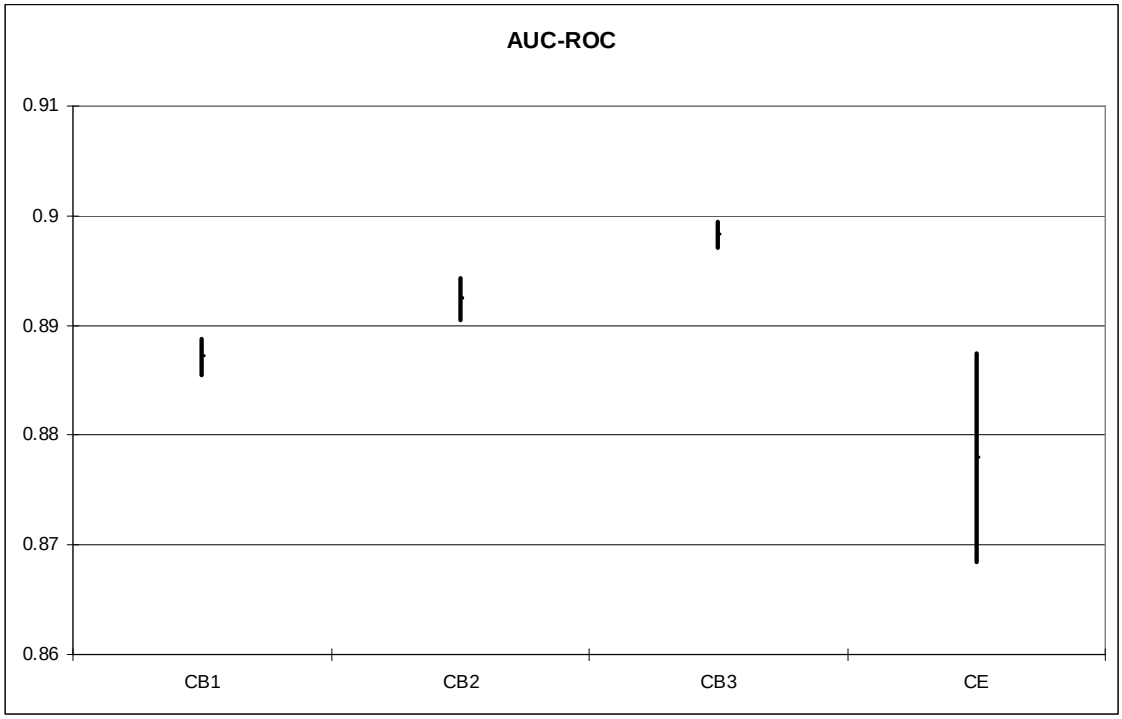

Figure 2. Area under the ROC curve, averaged over all experiments.

Each method is significantly different for the others with greater than 95% confidence (using a reverse bar-overlaps-bar test [16]). CB3 performs the best, having the highest AUC-ROC, followed by CB2 and CB1. These ranges are nearly identical before and after calibration.

Figures 3 and 4 show 95% confidence intervals for measured calibration mean-squared error (CMSE), or the average squared difference between network outputs and measured posteriors, before and after the calibration of test pattern outputs.

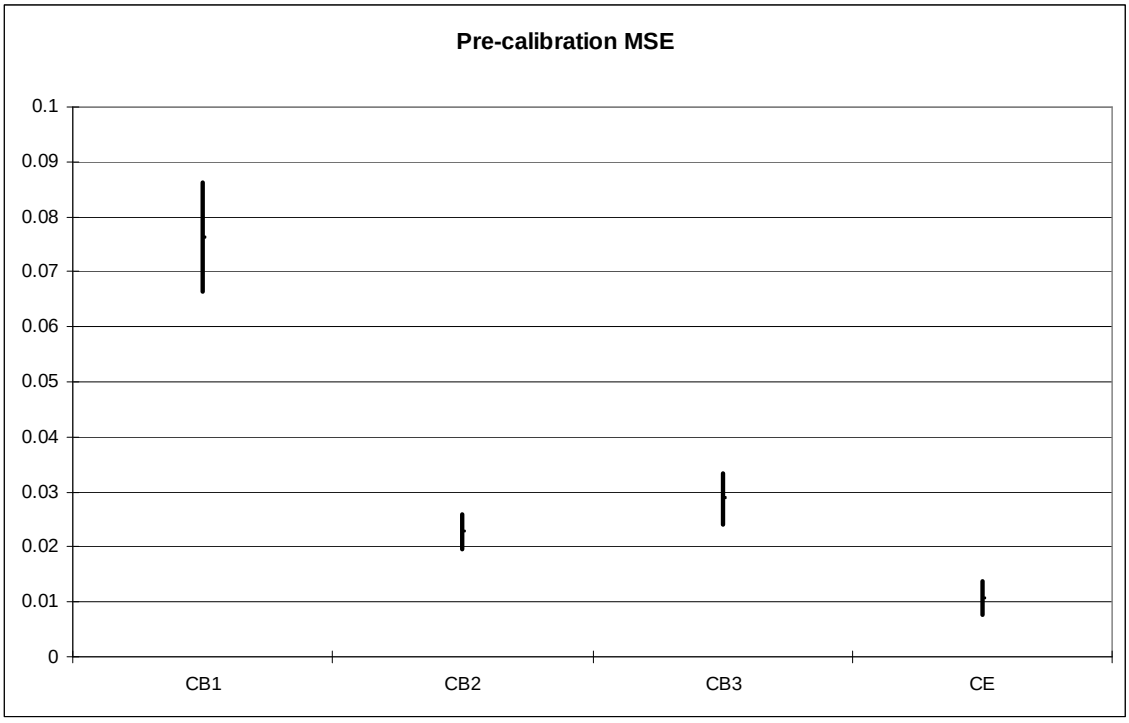

Figure 3. Pre-calibration MSE, averaged over all experiments.

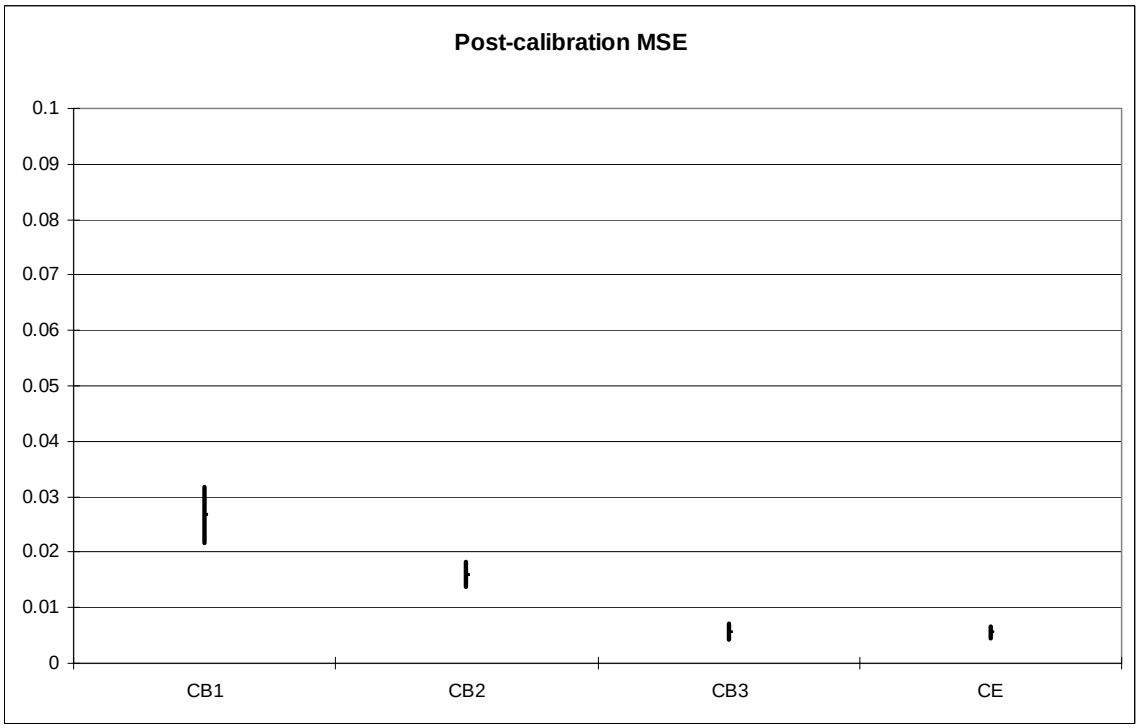

Figure 4. Post-calibration MSE, averaged over all experiments.

Before calibration, CB1 has a high CMSE of 0.076, which equates to a more than 26% inaccuracy in the posterior probability estimate on average. CE has the best CMSE at about 0.01, which is a 10% inaccuracy in posterior estimation. After calibration, the CMSE of all learning methods is significantly improved. CB1 has a post-calibration CMSE roughly equal to pre-calibration CMSE for CB2 and CB3. Calibrated CB3 has significantly lower CMSE than uncalibrated CE networks. Calibrated CB3 and CE networks have a nearly identical CMSE of about 0.0055, which translates into a 7.5% inaccuracy in posterior estimation, on average. Consequently, inaccuracy in posterior estimation for CB1 and CB3 has been decreased by more than half, and this inaccuracy is reduced by about 25% for CB2 and CE.

5 Conclusions

In this work, we applied the PL1 algorithm [9] to aid CB trained networks in outputting more accurate posterior probabilities. Empirical tests on eleven classification problems showed that using this approach to calibrate CB networks significantly reduces their calibration error. After calibration, CB1 becomes about as well-calibrated as CB2 and CB3 without calibration. Calibrated CB3 networks output probability estimates that are on par with calibrated CE networks. CB3 models have superior classification accuracy over CE networks before and after calibration, which suggests the use of calibrated CB3 networks for problem domains where high accuracy and accurate probability estimation is desired.

References

1. Niculescu-Mizil, A. & Caruana, R. (2005). Predicting Good Probabilities with Supervised Learning. Proceedings of the American Meteorology Conference (AMS2005), San Diego.

2. Rimer, M. & Martinez T. (2006). Classification-based Objective Functions. *Machine Learning*, *63:2*, 183-205.

3. Rimer, M. & Martinez, T. (2004). Softprop: Softmax Neural Network Backpropagation Learning. Proceedings of the *IEEE International Joint Conference on Neural Networks IJCNN'04*, 979-984.

4. Rimer, M. & Martinez T. (2006). CB3: An Adaptive Error Function for Backpropagation Training. *Neural Processing Letters*.

5. DeGroot, M. & Fienberg, S. (1982). The comparison and evaluation of forecasters. *Statistician*, 32, 12-22.

6. Platt, J. (1999). Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. Advances in Large Margin Classifiers (pp. 61-74).

7. Zadrozny, B. & Elkan, C. (2001). Obtaining calibrated probability estimates from decision trees and naïve Bayesian classifiers. *International Conference on Machine Learning*, pp. 609-616.

8. Zadrozny, B. & Elkan, C. (2002). Transforming classifier scores into accurate multiclass probability estimates. *Knowledge Discovery and Data Mining*, pp. 694-699.

9. Rimer, M & Martinez, T. (2007). Improving Posteriors with Point-wise Local Binning. Submitted to *Neural Processing Letters*.

10. Zadrozny, B. & Elkan, C. (2001). Learning and making decisions when costs and probabilities are both unknown. In *Proceedings of the Seventh International Conference on Knowledge Discovery and Data Mining*, pp. 204-213. ACM Press.

11. Ayer, M., Brunk, H., Ewing, G., Reid, W., & Silverman, E. (1955). An empirical distribution function for sampling with incomplete information. Annals of Mathematical Statistics, 5, 641–647.

12. Wilson, D. & Martinez, T. (1997). Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research* (JAIR), 1, pp. 1–34.

13. Blake, C. and Merz, C. (1998). UCI machine learning repository. http://www.ics.uci.edu/~mlearn/MLRepository.html. Irvine, CA: University of California, Department of Information and Computer Science.

14. Thimm, G. and Fiesler, E. (1997). High-order and multilayer perceptron initialization. IEEE Transactions on Neural Networks, 8:2, pp. 249-259.

15. Krogh, A. & Hertz, J. (1992). A simple weight decay can improve generalization. Advances in Neural Information Processing Systems (NIPS) 4, pp. 950-957, San Mateo, CA.

16. Minka, T. (2002). Judging significance from error bars. http://research.microsoft.com/~minka/papers/minka-errorbars.pdf.

Chapter 9

Speed Training: Improving the Rate of Backpropagation Learning through Stochastic Sample Presentation

Michael E. Rimer, Timothy L. Andersen and Tony R. Martinez

Brigham Young University
Computer Science Department
Provo, UT 84602, USA
{mrimer, tim}@axon.cs.byu.edu, martinez@cs.byu.edu

Abstract. Artificial neural networks provide an effective empirical predictive model for pattern classification. However, using complex neural networks to learn very large training sets is often problematic, imposing prohibitive time constraints on the training process. We present four practical methods for dramatically decreasing training time through dynamic stochastic sample presentation, a technique we call speed training. These methods are shown to be robust to retaining generalization accuracy over a diverse collection of real world data sets. In particular, the SET technique achieves a training speedup of 4278% on a large OCR database with no detectable loss in generalization.

1 Introduction

Artificial neural networks have received substantial attention as robust learning models for tasks including classification [5]. Much research has gone into improving their ability to generalize beyond the training data. Many factors play a role in their ability to learn, including network topology, learning algorithm, and the nature of the problem at hand.

In particular, the measure to which the training set represents the underlying distribution influences ultimate classification accuracy. Overfitting the training data is often detrimental to generalization. In theory, amassing an infinite training set would provide an exact measure of test accuracy (complete representation of the data distribution) and discourage overfitting. Hence, it is desirable to incorporate as large a training set as possible into the learning phase. However, training on very large data sets is problematic, as training time tends to increase more than linearly with the size of the training set [3]. The time required to converge on large data sets can be prohibitive. We provide four novel learning approaches that have shown to decrease training time by over an order of magnitude on very large data sets. Notably, the SET method achieves a training speedup of up to 4278% on the data tested with no detectable loss in generalization.

We give an overview of related work in section 2 and present four novel methods for speed training in section 3. Experiments are described in section 4. Results and analysis are given in section 5, followed by further work in section 6 and conclusion in section 7.

2 Related work

There have been many algorithms used to speed up the training of backpropagation neural networks, most of which are gradient descent "optimizing" algorithms. Two noteworthy approaches are QuickProp [2] and RProp [4]. QuickProp introduces a new error function, weight decay, and an alternative momentum equation. RProp uses an

exponentially adaptive step size for each parameter in the network [7]. These techniques allow quicker convergence. However, little research has involved how the nature and size of the training set affects the training speed and resultant generalization. Zhang [9] creates a training set by selecting only critical examples and then expands this set if necessary for proper convergence.

A simpler method of improving generalization through reducing overfitting is to provide a maximum error tolerance threshold, d_{max}, which is the smallest absolute output error to be back propagated [6]. In other words, for a given d_{max}, target value, t_j, and network output, o_j, no weight update occurs if the absolute error $| t_j - o_j | < d_{max}$. This threshold is arbitrarily chosen to represent a point at which a sample has been sufficiently approximated. With an error threshold, the network is permitted to converge with much smaller weights, translating to a reduction in overfitting.

When class data is unbalanced, techniques such as sub-sampling and re-sampling the training data can provide a way to reduce training time and improve generalization on the less represented classes [3]. Along with these techniques, Owens trains a committee of networks, each network learning from a distinct (balanced) subset of the training data. However, while this can improve training time and generalization, it results in a much more complex solution involving several networks instead of one. This technique's training time is reduced at the expense of testing time. In problem domains where a large amount of high-dimensional data is being classified, such solutions introduce a new problem by slowing down classification.

3 New approaches

Our proposed methods differ from Zhang's and Owens' in two main respects. First, we use a stochastic data selection mechanism based solely on the network's ability to learn the given data rather than statistical approaches focusing on feature redundancy. Second, whereas Zhang only adds more examples with time and does not allow them to be removed from the training set and Owens selectively determines the data as a step preliminary to training, we provide a temporally dynamic stochastic data inclusion mechanism that presents samples to the network according to present learning need. These differences are based on inferred feature correlation and data replication (identical or almost-identical samples) existing in artificial and real world data sets. Equivalent generalization is achieved in less time without increasing the complexity of the network.

Rather than initially selecting a small subset of the training data to present to the network during training, the network retains access to all data samples during the training process. Sample presentation is determined exclusively by the ability of the network to learn the data. These methods result in a large reduction in training time through selectively "pruning" correctly classified samples from the training set to exclude their (redundant) presentation to the network each epoch. In other words, only the samples currently affecting the learning process are presented. We refer to this method of reducing training time through selective sample presentation as *speed training*.

3.1 Error based presentation (Error Based)

Each sample from the training data is presented to the network during the first epoch. The output error of the net for each sample is recorded. In subsequent epochs, samples are stochastically presented to the network based on the previous amount of error, where the error translates to the probability of subsequent presentation. That is, the probability of a sample, x_i, being presented on the following epoch is equal to its absolute training error (a value between 0 and 1), or formally,

$$P(x_i) = \frac{|t_i - o|}{\|O\|} \quad (1)$$

where t_i is the sample's target value, o is the net output, and $\| O \|$ is a normalization factor describing the range of the activation function (e.g., 1 for a standard sigmoid function).

Therefore, samples already learned to a high degree of accuracy are rarely presented to the network, while samples with a high error are presented more often. This approach provides a mechanism to progressively speed up training as the network converges by bypassing unneeded examples (those that do little to update the network parameters) and focusing on the more difficult parts of the problem.

3.2 Stochastic presentation with error threshold (SET)

An error tolerance threshold, d_{max}, is incorporated so that network weights are only updated on samples that output an error greater than this threshold (as described in

Section 2). The probability of presenting a sample to the network is proportional to how close the sample is to overstepping the threshold. Formally,

$$P(x_i) = \begin{cases} \dfrac{|t_i - o|}{d_{\max}} & \text{if } |t_i - o| < d_{\max} \\ 1 & \text{otherwise} \end{cases}$$

This crudely equates to the probability the sample has of affecting the network parameters. Thus, samples with error far below the threshold will be seen rarely, while samples closer to the threshold will be seen often to maintain their correctness. This effectively bypasses samples that do not affect the performance of the network. Note that this method is more "conservative" than equation (1), skipping fewer samples on average.

3.3 Skip when correct (*n*-SKIP)

When the network classifies a sample correctly for *n* epochs, do not present it again for *n* epochs:

$$P(x_i) = \begin{cases} 0 & \text{if } (\text{last } n \text{ epochs correct}) \wedge \\ & \quad (\text{skipped less than last } n \text{ epochs}) \\ 1 & \text{otherwise} \end{cases}$$

where *n* is a parameter and "skipped" is when x_i is not presented during an epoch; we define "correct" as error within d_{max} for the experiments presented below. These parameters are determined by the problem at hand, and can include the network outputting in a range of values (e.g., above 0.6 or according to winner-take-all). The

intuition behind this method is that when the network incorrectly classifies a sample, it will probably incorrectly classify it again. Conversely, when the network is consistently correct on a sample, it will probably be correct again, and can therefore be skipped without adversely affecting the training process with high probability.

The tendency is that the more data there are, even when some samples are skipped, there will exist neighboring samples (closer to the decision surface) that are not skipped. This serves to keep the decision surface "in line" in the temporary absence of sample points. Re-including a sample after *n* epochs provides a quick check that the sample is still being classified correctly, and then if it is still correct it is skipped for another *n* epochs. The larger the value of *n*, the greater the speed up will be on large data sets, with the greater risk of samples falling "out of line" during their absence from several training epochs. This might result in greater deviation from standard training, but does not necessarily translate to a loss in generalization accuracy.

3.4 Stochastic presentation based on correctness history (Correct Ratio)

The probability of *not* presenting a sample is the ratio of the number of epochs for which it is correctly classified to the total number of epochs. We implement the probability of presentation through the formula

$$P(x_i) = 1 - \frac{\#\text{epochs correct}}{\#\text{epochs}} \qquad (2)$$

where *# epochs* includes the current epoch (so that there is always a chance for presentation). The more often a sample is classified correctly the less often it is presented. For our experiments, we did *not* consider a sample correctly classified when skipped. This conservatively avoids skipping samples more and more often with time without justification. Other variants are possible and are discussed in section 8.

3.5 Resource Requirements

For the above methods, additional resource requirements are modest, limited to $O(n)$ in both space and time over the number of samples.

4 Experiments

To measure the speedup achieved through these approaches as well as validate their integrity we tested them on various problem domains, from small toy problems to very large real world data sets.

4.1 Data sets

1. *4-AND.* A small "toy" problem (although it certainly can appear in real data) consisting of a 4-input *AND* function with 16 samples that completely cover the problem space.

2. *Breast cancer Wisconsin (bcw).* A medium-sized real world problem taken from the UCI Machine Learning Repository [8], consisting of nine input attributes, one binary output, and 549 patterns, randomly split into 439 training patterns and 110 test patterns.

3. *OCR.* A very large set of machine printed alphanumeric characters used for OCR. It consists of over 495,000 samples, randomly split into roughly 415,000 training samples and 80,000 test samples. For training, each sample was normalized onto an 8x8 grid, resulting in 64 inputs. We trained a network to distinguish each character, but for simplicity only the results for the character "a" (a typical category with about 15,000 samples) are presented here.

4.2 Learning parameters

We used fully connected feed-forward neural networks trained through standard on-line backpropagation (minimizing SSE) for all experiments. For learning the *4-AND, breast cancer,* and *OCR* problems the network contained a single hidden layer comprised of 4, 5, and 32 hidden nodes, respectively. Weights were initialized to uniform random values in the range [-0.3,0.3]. For a given data set, the same initial weight values were used for all training runs. We used a learning rate of 0.2, momentum of 0.5, and error threshold (d_{max}) of 0.1 in all experiments presented here. Training was stopped when no samples were classified incorrectly on *4-AND,* and when a maximum number of epochs was reached (1000 for *breast cancer* and 500 for *OCR*).

5 Results and analysis

Tables 1-3 display the results of each data set. *Epochs* is the number of epochs until convergence. *Samples* is the total number of samples presented to the network during the training run. *Time* is real training time in seconds. *% SpdUp* is the speedup in training time over the standard method, in percent. *Train* is the final training set accuracy (above 0.5 for positive samples, below 0.5 for negative samples) in percent. *Train MSE* is the mean squared error for the training set at convergence. *Test* is the test set accuracy in percent. *Test MSE* is the mean squared error for the test set. Best values for each column are in italics.

The Error Based presentation technique results in the greatest training speed up in general, from a 78% increase in speed on *breast cancer* to a 4487% speed up on *OCR*. Of all four methods, this one prunes samples most aggressively. This is at the expense of a slight decrease in generalization accuracy compared to standard sample presentation. Speed up on *breast cancer* is not as great as on other sets because the MSE is higher on this data set. Higher average error causes samples to be presented more often during Error Based presentation.

Table 1. Results on *4-AND*.

Method	Epochs	Samples	Time	% SpdUp	Train	Train MSE	Test	Test MSE
Standard	1499	23984	0.047	N/A	*100.0*	*0.0313*		
Error Based	*559*	*3126*	*0.016*	*193.75*	*100.0*	0.0945		
SET	1495	12539	0.032	46.88	*100.0*	*0.0313*		
3-SKIP	1165	7122	0.032	46.88	*100.0*	0.0609	N/A	N/A
6-SKIP	1325	8289	0.032	46.88	*100.0*	0.0409		
9-SKIP	1464	9333	0.032	46.88	*100.0*	0.0326		
Correct Ratio	1502	11092	0.032	46.88	*100.0*	*0.0313*		

Table 2. Results on *bcw*.

Method	*Epochs*	Samples	Time	% SpdUp	Train	Train MSE	Test	Test MSE
Standard		439000	1.281	N/A	94.76	*0.0947*	*90.91*	0.1293
Error Based		137990	0.719	78.16	*97.04*	0.1076	88.18	0.1478
SET		201248	0.859	49.13	94.76	0.0949	*90.91*	*0.1291*
3-SKIP	1000	*84959*	*0.484*	*164.67*	94.76	0.1289	*90.91*	0.1611
6-SKIP		92423	0.515	148.74	94.99	0.1335	90.00	0.1726
9-SKIP		95770	0.531	141.24	95.22	0.1239	90.00	0.1618
Correct Ratio		120293	0.640	100.15	95.22	0.1129	90.00	0.1544

Table 3. Results on *OCR*.

Method	*Epochs*	Samples	Time	% SpdUp	Train	Train MSE	Test	Test MSE
Standard		207100000	8527.946	N/A	99.99	*0.0002*	99.96	0.0006
Error Based		*939790*	*185.898*	*4487.43*	99.96	0.0011	99.93	0.0014
SET		1188387	194.773	4278.40	*100.00*	*0.0002*	*99.97*	*0.0005*
3-SKIP		52760243	2312.724	268.74	*100.00*	*0.0002*	99.96	0.0006
6-SKIP	5 0 0	31710579	1401.750	508.38	*100.00*	*0.0002*	99.95	0.0006
9-SKIP		24262191	1114.810	664.97	*100.00*	*0.0002*	99.96	0.0006
12-SKIP		20566468	942.520	804.80	*100.00*	*0.0002*	99.96	0.0006
18-SKIP		18116504	854.524	897.98	*100.00*	*0.0002*	*99.97*	*0.0005*
24-SKIP		18161186	857.508	894.50	*100.00*	*0.0002*	99.96	0.0006
Correct Ratio		4378508	328.290	2497.69	99.99	0.0005	99.94	0.0010

SET proves superior in terms of accuracy, generalizing equally well or better than standard training on all three data sets. It is more conservative than Error Based in choosing what samples to exclude, hence yields slightly slower training. It still improves training time by 4278% on *OCR*. In other words, training on this large data set is

performed in less than 2.3% of the standard time required. This translates to a drop in training time over one-and-a-half orders of magnitude, from hours to minutes (see Figures 1 and 2).

All variants of *n*-SKIP produced roughly equivalent results in generalization compared to standard training. They achieve a speed up roughly proportional to their *n* factor on large data sets. On fewer data, smaller *n* perform better. 3-SKIP learns *breast cancer* the quickest of all methods tested. 18-SKIP generalizes as well as SET on *OCR*, although it does not display as marked a decrease in training time (since 18 full epochs must occur before any samples are pruned).

Correct Ratio achieves higher accuracy and is faster on *breast cancer* than Error Based, although it is 76.6% slower on *OCR*. It is only slightly worse in generalizing than standard training. Its training time is roughly the median over all four methods on these data sets. As training continues, this technique tends to prune more and more samples. The percent of samples pruned per epoch is equivalent to the training set accuracy in the limit.

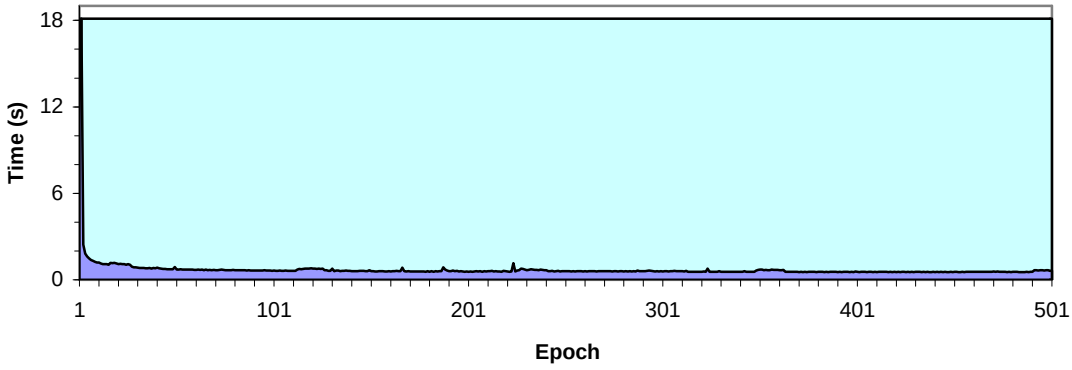

Figure 1. Training time per epoch (log scale) on *OCR* with SET (darker) vs. standard training.

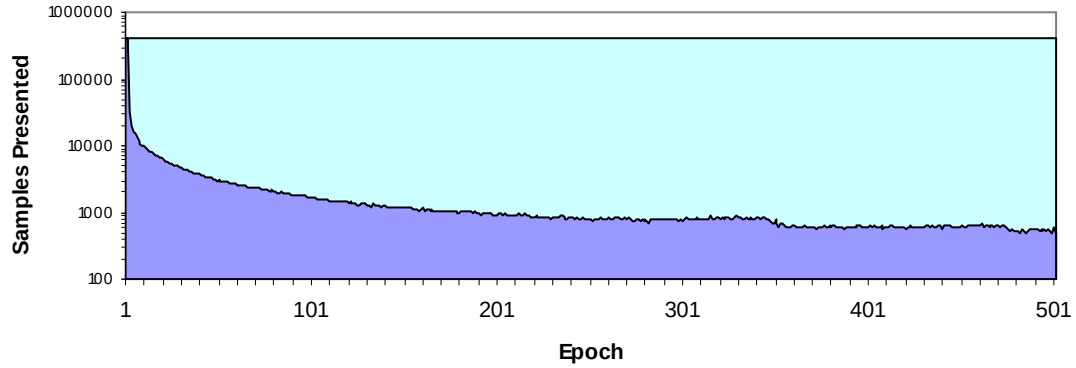

Figure 2. Samples presented per epoch (log scale) on *OCR* with SET (darker) vs. standard training.

6 Further Work

Further efforts will combine speed training with other "optimized" backpropagation algorithms (e.g., Quickprop and RProp). Together, it is conceivable that they will speed up convergence as well as reduce time spent per epoch in sample presentation.

Extending speed training to other iterative learning models, where the effectiveness or need of sample presentation varies over time, will also be studied. In particular, speed training will be tested with *batch* learning, where training time is very slow and epoch speed up is extremely desirable.

In addition to speeding up training, presenting samples with the most error more often may in general discourage overfitting. As proposed in [1], generalization is affected most by the size of the network parameters. When learning continues until weight saturation, generalization can be compromised. Excluding well-learned samples from further training can be a mechanism for keeping weights small, thereby improving generalization over techniques that saturate weight parameters. The usefulness of this principle will be investigated.

Several variations exist on the four methods proposed here. For example, when a sample is excluded from presentation on a given epoch, the probability that it will be presented in subsequent epochs can be gradually increased by a nominal value. This provides a more conservative approach to stochastic data exclusion, not allowing samples to be removed from training for too long.

Similarly, the way skipped samples affect sample presentation probability in Correct Ratio can be incorporated by extending equation (2) as follows:

$$P(x_i) = 1 - \frac{\#\text{epochs correct} + \alpha(\#\text{epochs skipped})}{\#\text{epochs}}$$

where α, ranging from zero to one, provides a pruning "aggressiveness" factor. For α approaching zero, skipping a sample increases the probability of presentation in subsequent epochs. This conservative approach reflects our experiments conducted here. For α close to one, skipping a sample gradually reduces the probability of subsequent presentation, a more aggressive pruning model.

Another improvement is to automate the choosing of *n* in *n*-SKIP in order to reduce training time as much as possible without requiring repeat training runs. An extension to this would be to dynamically alter the value of *n* during the training process to encourage further speedup.

Furthermore, the value from which $P(x_i)$ is derived in Error Based, SET, and Correct Ratio speed training can be augmented by a scaling factor to provide more conservative or aggressive sample pruning. However, a non-linear function of error to $P(x_i)$ is more general and may prove more effective. Investigation of these modifications will be presented in future work.

In the experiments presented here, no parameter optimizations were performed; commonly used, standard parameter values were incorporated for learning rate, momentum and error threshold. Work will be done to observe the effect of modifying these parameters on the time and accuracy of these speed training techniques.

7 Conclusion

Speed training provides an alternative to standard sample presentation in neural network training. It is a viable solution to overcoming prohibitive training costs in learning very large data sets with complex networks, and is an alternative to techniques such as subsampling [3] to reduce training time. It has proven effective on a variety of data sets with vastly different properties. Training time is reduced by roughly an order of magnitude and generalization is preserved.

A major weakness of standard backpropagation neural network learning is its slow training speed. Any of the proposed stochastic sample presentation schemes are appropriate if rapid training speeds are required while a very minimal drop in accuracy is acceptable. If accuracy is paramount, then conservative sample exclusion techniques, such as SET, provide dramatic speedup with no detectable loss of accuracy.

References

[1] Bartlett, Peter L., "The Sample Complexity of Pattern Classification with Neural Networks: The Size of the Weights is More Important than the Size of the Network", *IEEE Trans. Inf. Theory*, 44(2), 1998, pp. 525-536.

[2] Fahlman, S.E., "Faster-learning Variations on Back-propagation: An Empirical Study", *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufmann.

[3] Owens, Aaron J., "Empirical Modeling of Very Large Data Sets Using Neural Networks", *International Joint Conference on Neural Networks 2000*, vol. 6, pp. 6302-10.

[4] Riedmiller, Martin and Braun, Heinrich, "A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm", *Proceedings of the IEEE Conference on Neural Networks*, San Francisco, 1993.

[5] Rumelhart, David E., Hinton, Geoffrey E. and Williams, Ronald J., *Learning Internal Representations by Error Propagation*, Institute for Cognitive Science, University of California, San Diego; La Jolla, CA, 1985.

[6] Schiffmann, W., Joost, M. and Werner, R., "Comparison of Optimized Backpropagation Algorithms", *Artificial Neural Networks*, European Symposium, Brussels, 1993.

[7] Schiffmann, W., Joost, M. and Werner, R., "Optimization of the Backpropagation Algorithm for Training Multilayer Perceptions", University of Koblenz: Institute of Physics, 1994.

[8] University of California, Irvine, Machine Learning Repository. http://www.ics.uci.edu/~mlearn/MLRepository.html

[9] Zhang, Byoung-Tak, "Accelerated Learning By Active Example Selection", *International Journal of Neural Systems*, 5(1), Germany, 1994, pp. 6775-79.

Chapter 10

Conclusion and future work

Three classification-based algorithms for ANN learning, CB1-3, have been presented and evaluated on several applications under a variety of learning conditions. CB1 was shown to perform significantly better than conventional SSE and CE error functions on several small- to medium-sized benchmark data sets, on a large *OCR* data set, and on the *TIDIGITS* corpus as part of a speech recognition system. CB2 was shown to have higher average classification accuracy than CB1, although the increase was not significant ($p = 0.05$). However, CB2 was observed to have significantly higher generalization when measuring the area under the ROC curve (AUC-ROC), lower variance in results and lower sum-squared error than CB1. CB3 was shown to be significantly superior to CB1, CB2, SSE and CE, with and without weight decay, on the data sets tested with respect to classification accuracy and AUC-ROC. CB3 was also shown to be most robust to the size of the network, learning parameters, and convergence criteria. Network models trained with CB1-3 were demonstrated to have lower average weight magnitude for each layer of the network than training with SSE or CE while converging in a comparable number of training epochs.

PL1, a point-wise local binning method for model calibration, was proposed and applied to calibrating ANNs and other machine learning models. It was shown to be more effective than isotonic regression, a state-of-the-art calibration method, in improving neural network posteriors. A subsequent study showed PL1 calibration of CB-trained

networks to be highly effective in reducing posterior estimation error. Calibrated CB3 networks were measured as being as precise as calibrated networks trained to optimize CE.

There are other ways to apply CB training techniques. Empirical tests showed that pattern misclassifications are not highly correlated among the various error functions. This observation invites study into the practical application of combining networks trained with various error functions into hybrid ensembles. It is expected that this will further improve generalization with respect to a variety of goodness metrics over both stand-alone models and homogeneous ensembles.

It is also worthwhile to study the efficacy of using more than one global objective function in the training of a network. One example of this is to train a model to optimize with respect to one metric and then continue optimizing with respect to another. CB-trained networks were shown to avoid pre-mature weight saturation and converge with significantly lower weight magnitudes than SSE- and CE-trained networks. In some cases, the trained network has weight magnitudes not much larger than the initial untrained network. We have considered first training a network with CB3, selecting the best model epoch with a validation set, and then training that model to optimize SSE or CE. Beginning SSE or CE training in a state of high accuracy and low network weights can preempt the possibility of converging to less-optimal local minima while retaining the possibility of further improving the network's accuracy. Preliminary results on benchmark data sets are encouraging.

Another conceivable method of training with more than one error function is to learn different portions of a problem domain with different cost functions. Local regions of a problem's feature space may be learned more accurately when optimizing with respect to one error function than another. In this case, it may be better to trade optimizing a single, global error function in favor of optimizing with respect to different error functions in local, distinct regions of the problem space. CB3 does this in some measure by altering the strength of the error signal backpropagated through the network based on the network's ability to learn training patterns, but more direct approaches may be useful in providing better human understanding of a problem.

Another way to divide a problem into separate learnable regions is semantically, by concept class. In multi-class problems, certain classes may better be learned by some model representations than by others. This would probably be most apparent in complex problems with a high number of classes, such as an OCR, document identification, or speech recognition system. Developing approaches for semantically grouping problem classes according to their ability to be learned when optimizing with respect to various error functions could prove useful to formulating more structured problem representations, thereby facilitating the implementation of mechanisms for more effective learning.

Appendix A

Optimal Artificial Neural Network Architecture Selection for Bagging

Timothy L. Andersen, Michael E. Rimer, and Tony R. Martinez

Brigham Young University
Computer Science Department
Provo, UT 84602, USA
{tim, mrimer}@axon.cs.byu.edu, martinez@cs.byu.edu

Abstract. This paper studies the performance of standard architecture selection strategies, such as cost/performance and CV based strategies, for voting methods such as bagging. It is shown that standard architecture selection strategies are not optimal for voting methods and tend to underestimate the complexity of the optimal network architecture, since they only examine the performance of the network on an individual basis and do not consider the correlation between responses from multiple networks.

1 Introduction

There are several well-known methods for combining the predictions of multiple classifiers in order to obtain a single prediction. These include Bayesian methods [16], bagging [6], boosting[13], and other voting methods [19]. However, little work has been done on the problem of model selection when using these methods. This paper examines the problem of selecting an appropriate neural network architecture when using bagging and other voting methods to combine the predictions of multiple neural networks. We

show that standard architecture selection strategies do not always select optimal neural network architectures for such methods.

Section 2 discusses voting methods and the problem of selecting an optimal network architecture for such methods. Section 3 discusses related work in the field of architecture selection. Section 4 gives experimental results, and section 5 gives the conclusion.

2 Architecture Selection for Voting Methods

Neural network architecture selection strategies studied in the literature have focused on choosing the single best performing architecture from a group of architectures, generally using some kind of cost/performance tradeoff or the performance of the network on a holdout set as the selection criteria. Under certain assumptions, these architecture selection criteria can be shown to be optimal. However, such performance measures are only optimal in the case where a single network is to be used as the final predictor, and are not optimal for the architecture selection problem when using bagging or other voting methods to combine the predictions of several neural networks. From a Bayesian standpoint, the optimal prediction is obtained by calculating a weighted average of all possible network architectures and all possible weight settings for those architectures, where each network is weighted by its posterior probability. From a purely Bayesian standpoint, any architecture selection strategy which chooses a single network

architecture using a cost/performance tradeoff is sub-optimal, since it entirely ignores a large number of possible architectures that could significantly impact the solution.

Obviously, the calculation of this weighted average is computationally infeasible; however, the optimal prediction can be approximated in a number of different ways. Bagging, which can be viewed as an approximation to the Bayes optimal solution, generates a prediction by calculating a weighted average of several predictors. With bagging, the weight is usually set to 1 for each predictor, which amounts to the assumption that all of the predictors are equally probable from a Bayesian standpoint. This assumption is not unreasonable since the predictors are often not likely to greatly differ in their posterior probabilities, and it may be difficult to accurately estimate the true, relative a-priori probabilities.

Bagging and other voting methods work best when the errors between the various predictors are uncorrelated, and the correct responses between the predictors are correlated. Generally speaking, very simple predictors tend to have both correlated errors and correlated correct responses. For example, one of the simplest ways to formulate a predictor is to always predict the majority class of the training set. Obviously, using multiple such predictors cannot increase classification accuracy, since the errors (and correct responses) of such predictors are 100 percent correlated. As the complexity of the predictors is increased, the correlation between the responses of the predictors tends to decrease. This is because with increasing complexity there is a corresponding increase in

the number of different solutions (minimum error for the training set) that the predictor can produce.

Since bagging and other voting methods work best when the correct responses between predictors are correlated and the incorrect responses are uncorrelated, when bagging or other voting methods are used to combine the results of multiple networks the goal for neural network architecture selection is to choose the network architecture that maximizes the correlation between multiple trained copies of the network when the networks are producing the correct response, and minimizes the correlation between the networks on incorrect responses. So, the network architecture which maximizes a cost/performance tradeoff, or even that performs the best on a holdout set, is not guaranteed to be the best architecture for bagging, since it does not examine this correlation.

There are a number of factors that can influence the choice of the appropriate network architecture for voting methods such as bagging. These include but are not limited to:

- Number of bagged predictors
- Number of training examples
- Underlying problem domain
- Idiosyncrasies of the training algorithm

For example, lowering the number of training examples is likely to require lowering the complexity of the network architecture in order to achieve optimal performance. It is also possible that increasing the number of predictors “in the bag” may allow for a corresponding increase in the complexity of the network architectures being bagged.

3 Related Work in Architecture Selection

There have been a number of different architecture selection strategies studied in the literature. These strategies are all ultimately based on either the use of a holdout set or a cost/performance tradeoff to determine the ‘optimal’ network architecture. These strategies include the following:

Network Construction Algorithms

The majority of network construction methods start from a very simple basis, usually one node, and add nodes and connections as needed in order to learn the training set. These strategies include Cascade Correlation [8], DNAL [4], Tiling [14], Extentron[3], Perceptron Cascade [7], the Tower and Inverted Pyramid algorithms [10], and DCN [17]. Other construction algorithms include Meiosis [11] and node splitting (Wynne-Jones 1992).

One of the drawbacks of most current MLP construction algorithms is that they do not have built in mechanisms to prevent the network from overlearning, rather treating this important subject as an afterthought. For example, Burgess states that "for good

generalization it is necessary to restrict the size of the network to match the task," [7] but no specific algorithm is presented on how to do so. Left uncontrolled, all of these methods will suffer from over learning, and so in some respects they do not avoid the architecture selection problem but must utilize some type of architecture selection strategy (such as CV or MDL based strategies) in an attempt to avoid over learning. This is due to the fact that, left uncontrolled, the network structure can grow to fit the training set data exactly. But with many problems the training data may contain noise that will cause the algorithm to perform worse if the noisy instances are memorized. Also, the network can grow to the point that the amount of training data is insufficient to properly constrain the network weights.

Early Stopping

Early stopping strategies [1,9,18,23] utilize overly complex network architectures. One of the main advantages of using a network that is more complex than is actually needed is that larger networks tend to have fewer local minima in the error surface. However, with a larger network there is a higher likelihood that over learning will occur. In other words, larger network architectures are more likely to converge to a lower training set error, but often tend to produce higher error on non-training examples. In order to avoid this, early stopping strategies try to determine when the network has been trained sufficiently to do well on the problem but has not yet over learned (or memorized) the training data. One way to do this is to occasionally test the performance of the network on a holdout set and stop training when the performance on the holdout set begins to degrade.

Cross Validation (CV)

CV is often used to select an optimal architecture from amongst a set of available network architectures. In a comparison of CV with two other MLP architecture selection strategies in a recent paper [20] CV was found to be the best at choosing the optimal network architecture, at least on the data sets tested. However, the comparison was based on only a single type of artificial data and did not look at any real world problem domains.

In a larger study CV was found to not perform well when selecting an optimal architecture from a large set of relatively similar architectures [2]. Several strategies are suggested which can be applied when using CV based MLP architecture selection to significantly improve the performance CV based architecture selection.

Weight Decay

Weight decay adds a penalty term to the error function that favors smaller weights [5, 12]. The rate of weight decay is often chosen by training several different networks with different rates of decay and then using CV to estimate which rate is optimal.

Network Pruning

Pruning techniques start with an overly large network and iteratively prune connections that are estimated to be unnecessary. CV is often used to assist in the estimation process. The pruning can take place during the training process or training cycles can be alternated

with pruning cycles. Pruning strategies include Optimal Brain Damage [21], Skeletonization [15], and Optimal Brain Surgeon [22].

4 Experiments and Results

Experiments were conducted several data sets in order to empirically determine the efficacy in cost/performance tradeoff and CV based methods in determining the optimal network architecture for bagging. The real world data sets were obtained from the UC Irvine machine learning database repository.

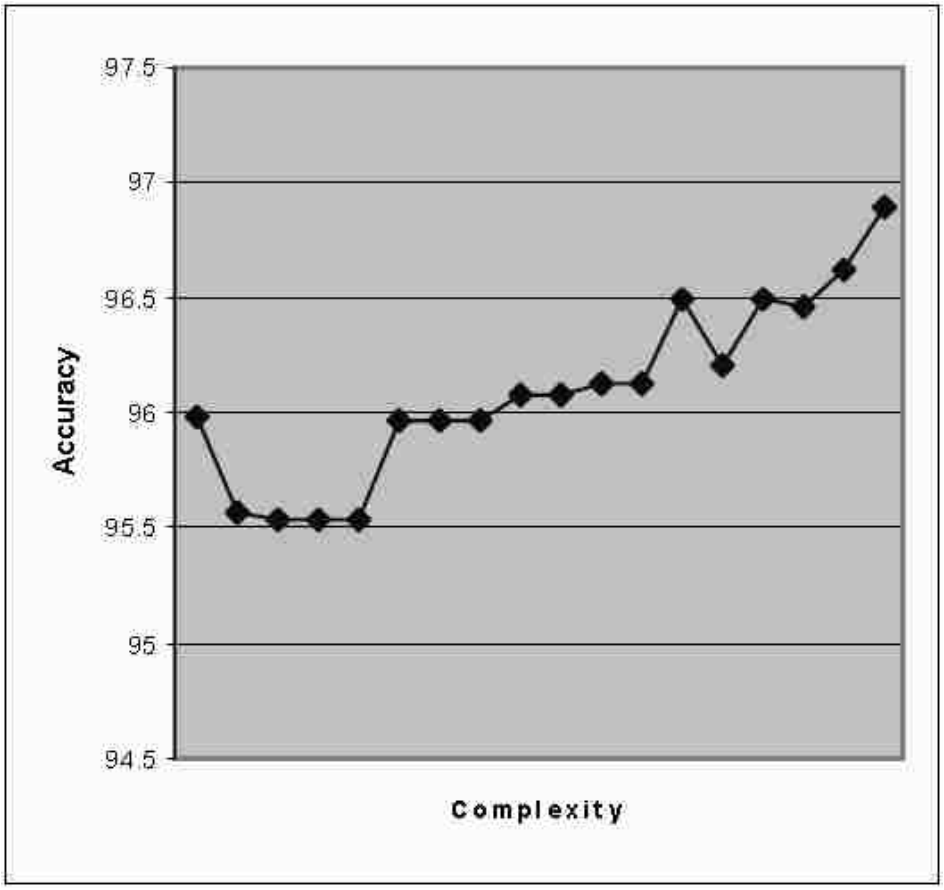

Figure 1. Breast Cancer Wisconsin and Bagging.

For the results reported in this paper the complexity level of the tested network architectures ranges from 2 to 20 hidden nodes arranged in a single hidden layer of a fully connected network. In order to determine the best architecture for bagging, 30 sets of network weights are trained for each complexity level in this range, and the performance of the 30 bagged networks is evaluated for each of the network architectures. The performance and complexity level of the best architecture for bagging is then compared against bagging's performance and complexity level using the network architecture chosen by Akaike's information-based criterion (AIC), and with the architecture selected by CV.

Figure 1 shows the test set results of the bagged networks for each of the network architectures tested on the Breast Cancer Wisconsin data set. This data set is interesting because it shows a significant general upward trend in test set accuracy as the complexity of the bagged networks is increased. However, there is not a significant upward trend in the test set scores of the networks taken individually (nor in the training set scores), as can be seen in figure 2. Because of this, architecture selection strategies which only examine the performance of the individual networks, such as most cost/performance measures and also CV based measures, are unlikely to find the optimal architecture for bagging for this particular problem. Indeed, for this particular problem the AIC measure chooses the simplest network architecture, which has a bagged network performance which is significantly worse than the best performance of the tested architectures (95.9% vs. 96.9% on the test set).

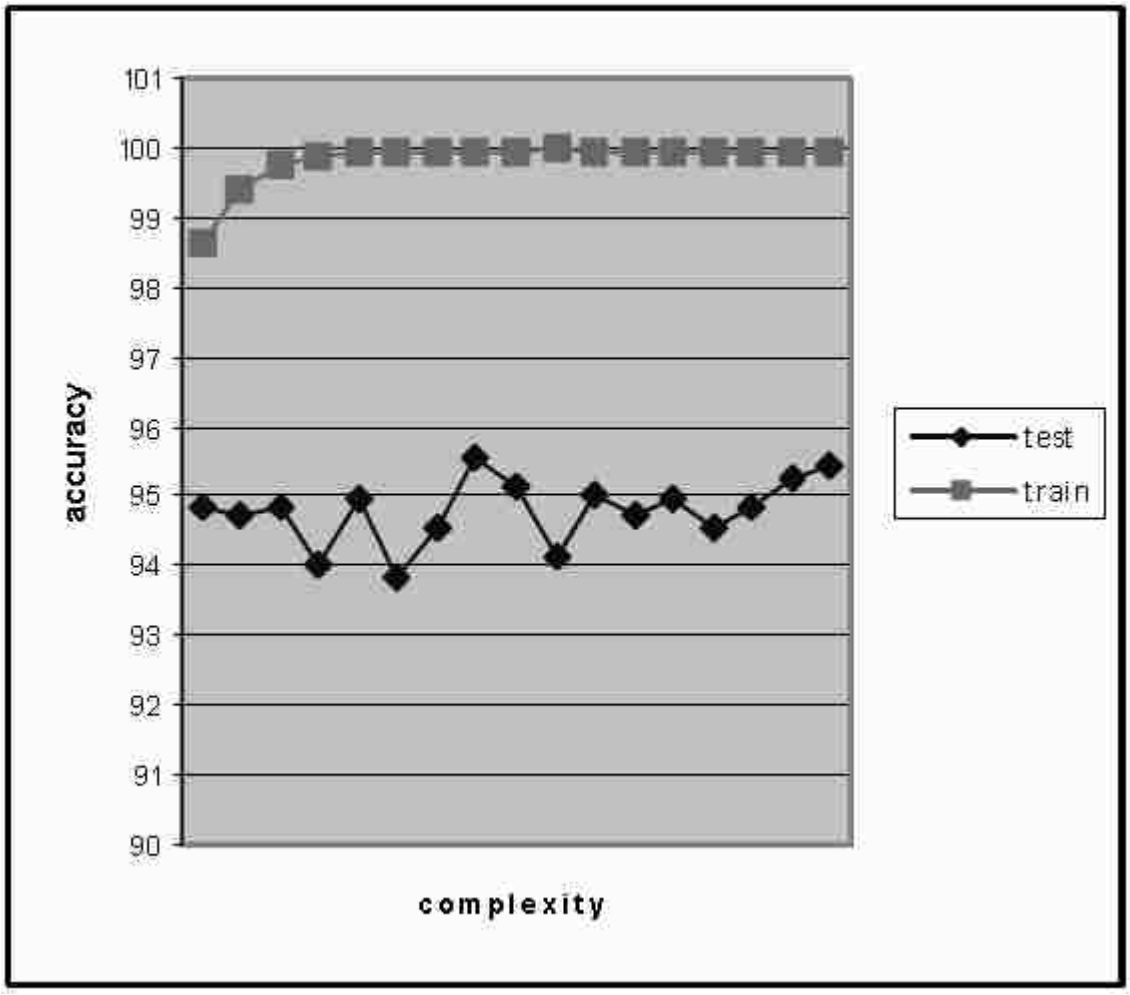

Figure 2. Breast Cancer Wisconsin - no Bagging.

Figure 3 shows the test set performance of AIC vs. CV for selecting the network architecture for bagging, and compares this against the 'optimal' network architecture for bagging. On average, the AIC criteria is significantly worse than using CV to choose an architecture for bagging, and both generally fail to pick the optimal network architecture for these problems.

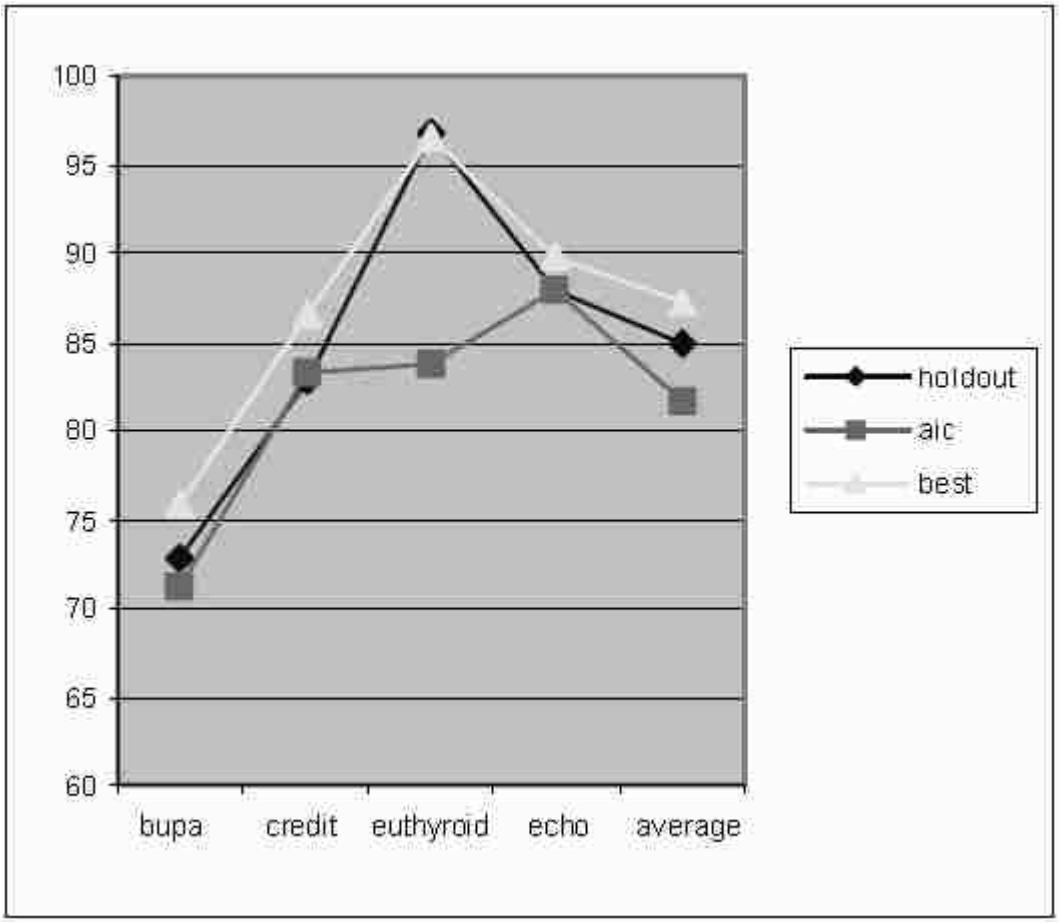

Figure 3. AIC vs. holdout vs. optimal test set accuracy.

Both AIC and CV significantly underestimate the complexity of the best architecture for bagging for these problems, as can be seen in figure 4, with AIC on average choosing a network with 4 hidden nodes and CV choosing an architecture with 6 hidden nodes, with the optimal network architecture for bagging containing (on average) 14 hidden nodes.

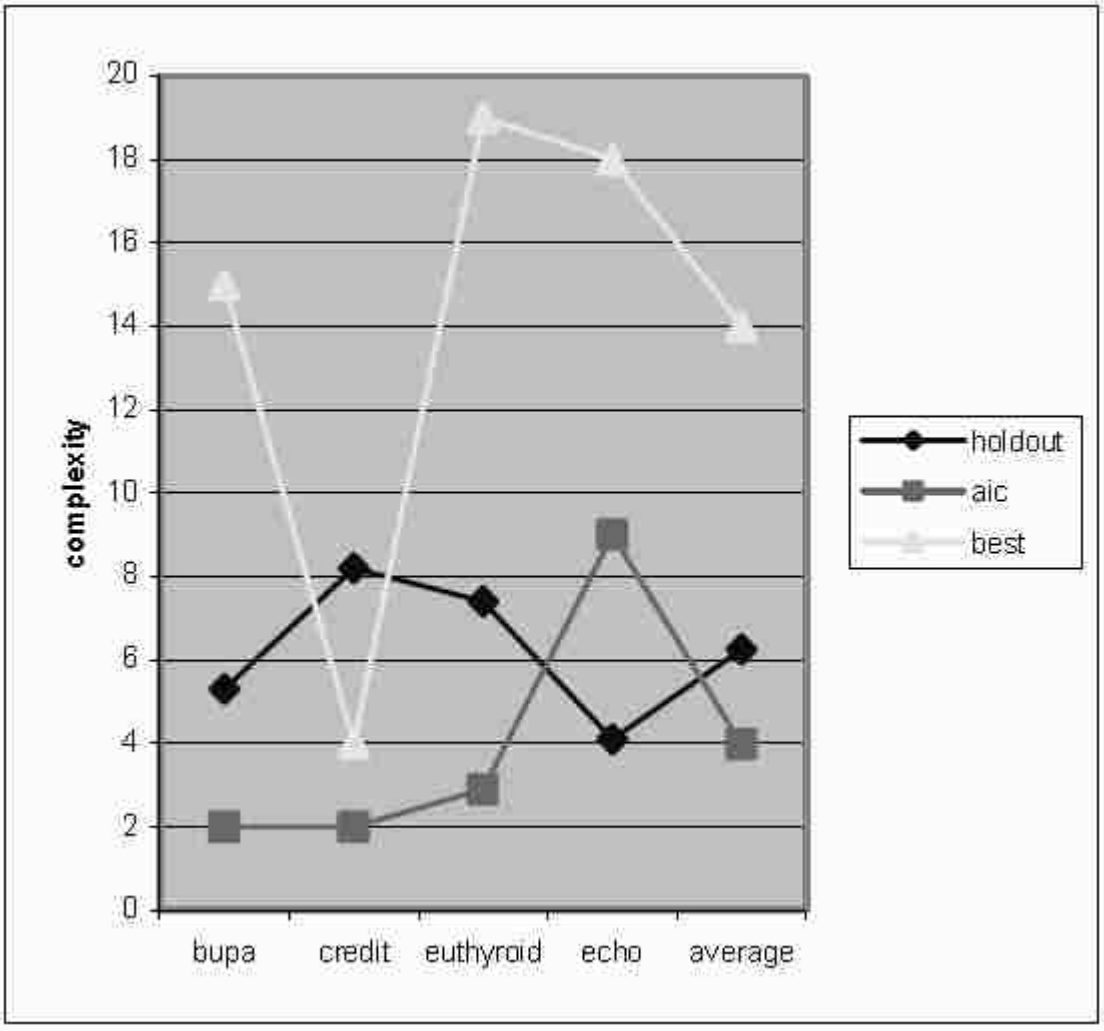

Figure 4. Average complexity of chosen architecture for each problem.

5 Conclusion

The experimental results show that, for the problems tested in this paper, the optimal network architecture for bagging (and by extension other voting methods) is more complex than the network architecture chosen by cost/performance tradeoff methods such as MML and MDL, and also more complex than the network architecture chosen by CV based methods which only examine the performance of individual networks. We have

argued that this empirical result will hold for most learning problems, since these strategies are only designed to identify the optimal network architecture if a single network will be used as the final predictor. When multiple networks are combined using a voting method, then these strategies tend to underestimate the complexity of the optimal network architecture since they cannot estimate the degree to which the responses of the different network architectures will be correlated, and this estimate is critical in the determination of the optimal network architecture for voting methods.

The factors which may affect the optimal complexity for bagging and other voting based methods include the number predictors that will be voted, the number of training examples, the underlying problem domain, and idiosyncrasies of the training algorithm. Future work will focus on studying the effects of each of these factors, as well as developing a systematic methodology for selecting the optimal network architecture for voting methods.

References

[1] Amari, S, Murata, N., Muller, K., Finke, M. & Hua Yang, H. 1997. Asymptotic Statistical Theory of Overtraining and Cross Validation. *IEEE Transactions on Neural Networks*, vol. 8, no 5, pp. 985-996, September 1997.

[2] Andersen, T. & Martinez, T. 1999. Cross Validation and MLP Architecture Selection. Proceedings of the *International Joint Conference on Neural Networks (IJCNN '99)*.

[3] Baffes, P. & Zelle, J. 1992. Growing Layers of Perceptrons: Introducing the Extentron Algorithm. Proceedings of the *International Joint Conference on Neural Networks (IJCNN '92)*, pp. 392-397, Baltimore, Maryland, June 1992.

[4] Bartlett, E. 1994. Dynamic Node Architecture Learning: An Information Theoretic Approach. *Neural Networks*, vol. 7, no 1, pp. 129-140.

[5] Bartlett, P. L. 1997. For valid generalization, the size of the weights is more important than the size of the network. *Advances in Neural Information Processing Systems 9*, pp. 134-140. Cambridge, MA: The MIT Press.

[6] Breiman, L. 1996. Bagging Predictors. *Machine Learning* 24, 123-140.

[7] Neil, B. 1994. A Constructive Algorithm that Converges for Real-Valued Input Patterns. *International Journal of Neural Systems*, vol. 5l, no 1, pp. 59-66, March, 1994.

[8] Fahlman, S. E. & Lebiere, C. 1990. The Cascade Correlation Learning Architecture. *Neural Information Processing Systems 2*, D. S. Touretzsky, ed. Morgan Kaufman, pp. 524-532.

[9] Finnof, W., Hergert, F. & Zimmermann, H. 1993. Improving model selection by nonconvergent methods. *Neural Networks*, 6, 771-783.

[10] Gallant, S. I. 1986. Three constructive algorithms for network learning. Proc. 8th *Annual Conference of Cognitive Science Soc*, pp. 652-660, August 1986.

[11] Hanson, S. 1990. Meiosis networks. In Advances in Neural Information Processing Systems, D. S. Touretzky, editor, pp. 533-541. Morgan Kaufman, San Mateo.

[12] Krogh, A. & Hertz, J. 1992. A Simple Weight Decay Can Improve Generalization. In Advances in Neural Information Processing Systems, Moody, J; Hanson S; and Lippmann, R eds, vol 4, pp. 950-957. San Mateo, CA: Morgan Kauffmann publishers.

[13] Maclin, R. & Opitz, D. 1997. An empirical evaluation of bagging and boosting. The Fourteenth National Conference on Artificial Intelligence.

[14] Mezard, M. & Nadal, J. P. 1989. Learning in feedforward layered networks: The tiling algorithm. *Journal of Physics* A, 22:2191-2203.

[15] Mozer, M. C. & Smolensky, P. 1988. Skeletonization: A technique for trimming the fat from a network via relevance assessment. Advances in *Neural Information Processing Systems 1*. D. S. Touretzky, Ed.. Denver 1988. pp. 107-115.

[16] Neal, R. M. (1996), Bayesian Learning for Neural Networks 118, New York: Springer-Verlag.

[17] Romaniuk, S. & Hall, L. 1993. Divide and Conquer Neural Networks. *Neural Networks*, vol 6, pp. 1105-1116.

[18] Sarle, W. S. 1995. Stopped Training and Other Remedies for Overfitting. Proceedings of the 27th *Symposium on the Interface of Computing Science and Statistics*, pp. 352-360.

[19] Schapire, R., Freund, Y., Bartlett, P. & Lee, W. S. 1997. Boosting the margin: A new explanation for the effectiveness of voting methods. Proceedings of the *Fourteenth International Conference of Machine Learning*.

[20] Schenker, B & Agarwal, M. 1996. Cross-validated structure selection for neural networks. *Computers Chem. Eng*, vol. 20, no 2, pp. 175-186.

[21] Solla S., Le Cun, Y. & Denker, J. 1990. Optimal Brain Damage. In Advances in *Neural Information Processing Systems* (NIPS) 2, pp. 598-605, D. S. Touretzky, editor, San Mateo, Morgan Kaufmann Publishers Inc.

[22] Stork D. & Hassibi, B. 1993. Second order derivatives for network pruning: Optimal Brain Surgeon. In Advances in *Neural Information Processing Systems*. (NIPS) 5, pp. 164-171, T. J. Sejnowski G. E. Hinton and D. S. Touretzky, editors, San Mateo, Morgan Kaufmann Publishers Inc.

[23] Wang, C., Venkatesh, S.S. & Judd, J.S. 1994. Optimal Stopping and Effective Machine Complexity in Learning. *NIPS* 6, pp. 303-310.

Appendix B

Network Simplification through Oracle Learning

Joshua Menke, Adam Peterson, Mike Rimer, Tony Martinez
Computer Science Department, Brigham Young University, Provo, UT, 84602
Email: {josh, adam, mrimer}@axon.cs.byu.edu, martinez@cs.byu.edu

Abstract. Often the best artificial neural network to solve a real world problem is relatively complex. However, with the growing popularity of smaller computing devices (handheld computers, cellular telephones, automobile interfaces, etc.), there is a need for simpler models with comparable accuracy. The following research presents evidence that using a larger model as an oracle to train a smaller model on unlabeled data results in 1) a simpler acceptable model and 2) improved results over standard training methods on a similarly sized smaller model. On automated spoken digit recognition, oracle learning resulted in an artificial neural network of half the size that 1) maintained comparable accuracy to the larger neural network, and 2) obtained up to a 25% decrease in error over standard training methods.

1 Introduction

As Le Cun, Denker, and Solla observed in [1], often the best artificial neural network (ANN) to solve a real-world problem is relatively complex. They point to the large ANNs Waibel used for phoneme recognition in [2] and that of Le Cun et al. with handwritten character recognition in [3]. “As applications become more complex, the networks will presumably become even larger and more structured.” [1] The growing complexity of

neural networks in real-world applications presents a problem when using them in environments with less available memory and processing power (i.e. embedded systems like handheld computers, cellular telephones, etc.). Therefore, there is a demand to create smaller, faster, neural networks that still maintain a similar degree of accuracy. The oracle learning solution involves using the most accurate available model as an oracle to train a smaller model. We propose that oracle learning will result in simpler models that 1) have accuracy comparable to their oracles, and 2) have improved results over standard training methods for the same sized model. For the following experiment, simple feed-forward single-hidden layer ANNs were used as both the oracle and the *oracle-trained network* (OTN). We propose the use of the following nomenclature for classifying OTNs within this paper:

OTN (*n* *m*)

Reads "an OTN approximating an *n* hidden node ANN with an *m* hidden node ANN." For example:

OTN (200 100)

Reads "an OTN approximating a 200 hidden node ANN with a 100 hidden node ANN." The rest of the paper describes oracle learning in terms of ANNs since the experiments deal solely with ANNs. We refer to the oracle as an *oracle ANN* (which is no different than a standard ANN, it is just used as an oracle).

One of the advantages of using an ANN as an oracle is the ability to use unlabeled training data to train smaller ANNs. In speech recognition, for example, there are more than enough data, but it is difficult and expensive to hand label them at the phoneme level. However, if an oracle ANN exists, the smaller ANN can theoretically request as many labeled data points as is necessary to best approximate the larger or oracle ANN.

The idea of approximating a more complex model is not entirely new. Domingos used Quinlan's C4.5 decision tree approach from [4] in [5] to approximate a bagging ensemble and Zeng and Martinez used an ANN in [6] to approximate a similar ensemble (both using the bagging algorithm Breimen proposed in [7]). Craven and Shevlik used a similar approximating method to extract rules [8] and trees [9] from ANNs. Domingos and Craven and Shevlik used their ensembles to generate training data where the targets were represented as either being the correct class or not. Zeng and Martinez used a target vector containing the exact probabilities output by the ensemble for each class. The following research also used vectored targets similar to Zeng and Martinez since Zeng's results supported the hypothesis that vectored targets "capture richer information about the decision making process . . ." [6].

While, previous research has focused on either extracting information from neural networks [8,9], or using statistically generated data [5,6] for training, the novel approach we propose in this paper is to use the approximated network as an oracle. The next section explains the details of the oracle learning process.

2 Oracle Learning

Oracle learning involves the following 3 steps:

A. Oracle Preparing

B. Data Labeling

C. Oracle Learning

2.1 Oracle Preparing

The primary component in oracle learning is the oracle itself. Since the accuracy of the oracle ANN directly influences the performance of the final, simpler ANN, the oracle should be the most accurate classifier available, regardless of complexity (number of hidden nodes). The only requirement is that the number and type of the inputs and the outputs of each ANN (the oracle and the OTN) be the same.

2.2 Data Labeling

The main step in oracle learning is to use the oracle ANN to create a very large training set for the OTN to use. Fortunately the training set does not have to be pre-labeled since the OTN only needs the oracle ANN's outputs for a given input. Therefore the training set can consist of as many data points as there are available, including unlabeled points.

The key to the success of oracle learning is to obtain as much data as possible that ideally fit the distribution of the problem. There are several ways to approach this. In [6], Zeng

and Martinez use the statistical distribution of the training set to create data. Another approach is to add random jitter to the training set, again following its distribution. The easiest way to fit the distribution is to have more unlabeled *real* data. In many problems, like ASR, there are more than enough unlabeled data. Other problems where there are plenty of unlabeled data include intelligent web document classifying, optical character recognition, and any other problem where gathering the data is far easier than labeling them. The oracle ANN can label as much of the data as necessary to train the OTN at the phoneme level. Therefore, the OTN has access to an arbitrary amount of ideally distributed training data.

In detail, this step must create a target vector $\boldsymbol{t}$ for each input vector $\boldsymbol{x}$ where each t_i in t_1 . . . t_n (n being the number of output nodes) is equal to the oracle ANN's activation of output i given $\boldsymbol{x}$. Then, the final oracle learning data point contains both $\boldsymbol{x}$ and $\boldsymbol{t}$. In order to create the points, each available pattern $\boldsymbol{x}$ (labeled or not, but not including a small labeled subset for testing) is presented as an input to the oracle which then returns the output vector $\boldsymbol{t}$. The OTN's training set then consists of every $\boldsymbol{x}$ paired with its corresponding $\boldsymbol{t}$.

As an example, the following two vectors represent the target vectors for a given input. The first vector is a standard 0-1 encoded target where the 4th class is the correct one. The second is more representative of an ANN output vector (the oracle for the following experiments) where the outputs are between 0 and 1, and the 4th class is still the highest.

(1) <0,0,0,1,0>

(2) <0.27, 0.34, 0.45, 0.89, 0.29>

Now suppose the OTN outputs the following vector:

(3) <0.19, 0.43, 0.3, 0.77, 0.04>

The standard error would simply be the difference between the target vector in (1) and the output vector in (3) which is:

(4) <-0.19, -0.43, -0.3, 0.23, -0.04>.

Whereas the oracle-trained error would be the difference between the target vector in (2) and the output in (3):

(5) <0.08, -0.09, 0.15, 0.12, 0.25>

Notice the oracle-trained error in (5) is on average lower than the standard error in (4), and therefore the OTN learns a function that may be easier for standard back-propagation.

Once again, Zeng and Martinez found the use of vectored targets to give improved accuracy over using standard targets in [6].

2.3 Oracle Learning

For the final step, the OTN is trained using the data generated in step 2, making sure to utilize the targets exactly as presented in the target vector. The OTN must interpret each element of each target vector as the correct output activation for the output node it represents given the input paired with it, hence the ANN's learning algorithm may need to be modified depending on how it handles targets. For most ANNs, classification targets are encoded in binary with the correct class as 1 and all others as 0 and hence the error is generally computed as $\{0 \mid 1\} - o_i$ where o_i represents the output of node i. With oracle learning, the error would instead be the $t_i - o_i$ where, as stated above, t_i is the ith element of the target vector $\boldsymbol{t}$ paired with the input $\boldsymbol{x}$. The outputs of the OTN will approach the target vectors of the oracle ANN on each data point as training continues.

3 Experiment

One of the most popular applications for smaller computing devices (i.e. hand held organizers, cellular phones, etc.) and other embedded devices is automated speech recognition (ASR). Since the interfaces are limited in smaller devices, being able to recognize speech allows the user to more efficiently enter data. Given the demand and usefulness of speech recognition in systems lacking in memory and processing power, there is a demand for simpler ASR engines capable of achieving acceptable accuracy.

Hence the following experiments seek to reduce the complexity of our current ASR engine—or more specifically, the phoneme classifying ANN portion of the engine.

The following experiments use data from the unlabeled TI digit corpus [10] for testing the ability of the oracle ANN to create accurate phoneme level labels for the OTN. The corpus was partitioned into a training set of 15,322 utterances (3,000,000 phonemes), and a test set of 1000 utterances. A small subset of the training corpus consisting of around 40,000 phonemes was labeled at the phoneme level for training the oracle ANN. The inputs are the first 13 mel cepstral coefficients and their derivatives in a 16 ms frame extracted from wav files every 10 ms (overlapping).

It is important to mention the fact that the final measure of accuracy is performed at the word and utterance levels, not the phoneme level. In general, word and sentence accuracies are more significant in speech recognition and do not always directly correlate with phoneme accuracy. It depends on the decoding technique and / or speech model used to build phonemes into words. In fact, in preliminary experiments, the standard trained networks always had slightly better phoneme accuracies than the OTNs (for any size).

Figure 1 diagrams the basics of the ASR engine used for the experiments. The mel cepstral coefficients are fed into the ANN and the ANN phoneme outputs are decoded into words. Both the oracle ANN and the OTN are used as the neural network recognizer part of the engine when determining word and utterance accuracy.

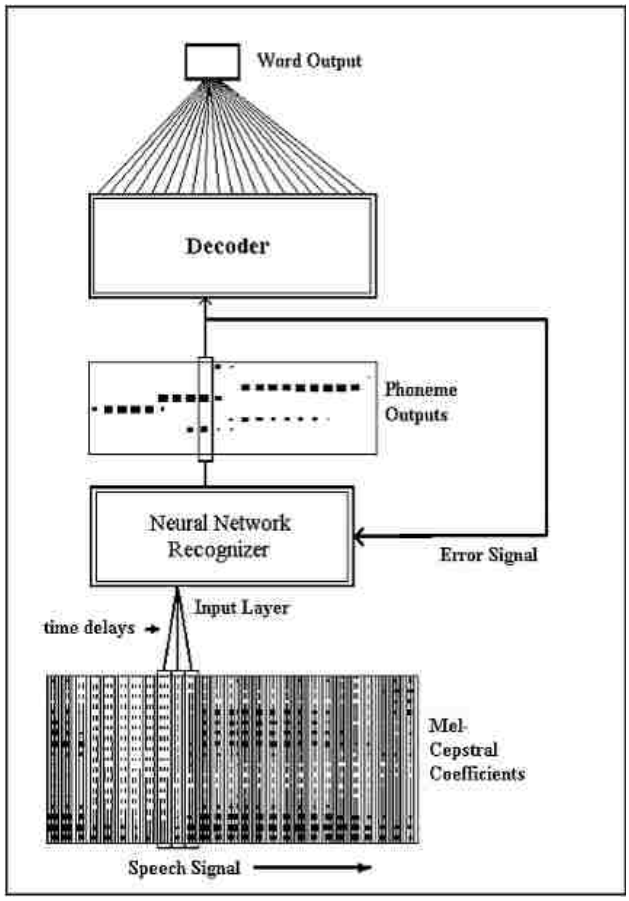

Figure 1. The basic ASR Engine.

A) Obtaining the Oracles

The ASR engine's standard neural network recognizer is a 200 hidden node standard back-propagation-trained feed-forward network that has been tuned and optimized over time. In the following experiment, the ANN is trained directly on the phoneme labeled training data, storing the ANN weight configurations for future testing. Although the ANN most accurate on the test set (words and utterances) was chosen as the oracle ANN, any one of them was sufficient to validate oracle learning as long as the OTN achieves

similar accuracy. We chose to use the most accurate ANN in order to create the most accurate OTN.

B) Labeling the Data Set

For the next step a large training set was created from the unlabeled data. The entire 15,000+ utterance training set was used to create a new training set consisting of the inputs from the old set combined with the target vectors from each oracle (one data set for each oracle), acquiring the target vectors as explained in section 2.2 (from the oracle ANN's outputs). In detail, oracle learning presents the oracle with an input pattern and then saves the activations of each output node for that input as a vector. The new OTN's training vector then consists of the original input and the new target vector.

C) Oracle Learning

Finally, the large OTN training set created in B is used to train an ANN half the size of the oracle (100 hidden nodes) using vectored targets instead of 0-1 targets according to the method described in section 2.3. For a given training pattern, the error back-propagated was set to the difference between the oracle ANN's output node activation and the OTN's output or $t_i - o_i$ where t_i is the oracle's output for class i and o_i is the output of the OTN net on class i.

To measure the effectiveness of oracle learning during the training phase, several metrics were used: the mean error with respect to the target vector, accuracy compared to the oracle ANN, and the top 100 OTN outputs compared the top 100 oracle ANN outputs. The general trend during training was for each of the metrics to improve, however, contrary to intuition, the best OTNs did not have the best values according to our metrics. It would be intuitive to believe the ANN with the least error with respect to the oracle would perform most like the oracle and hence have the best overall accuracy, but it did not. We hypothesize the reason was the phoneme-to-word decoding module did better with networks better arranging the ordering (from highest to lowest) of the output activation levels, regardless of the single highest output of the oracle ANN. The decoder considers more than just the top output, so where the next several outputs are ordered correctly, better word accuracy results. Therefore, even though one network may be more likely to have the same highest scoring phoneme as the oracle, the final ordering of the probabilities is better in a network with slightly a worse overall accuracy against the oracle.

A standard 100 hidden node network was also trained in order to compare it to the oracle learning 100 hidden node OTN (200 100).

After every oracle learning epoch, word and utterance accuracies were gathered and the respective OTN weights saved. The weights of the most accurate epoch were chosen as the best OTN of that particular oracle learning run.

4 Results and Analysis

Table 1 reports the accuracy for each of the mentioned ANNs on the test set (the standard back-propagation-trained 200-hidden node ANN used as the oracle, the OTN (200 100) and the 100 hidden node standard net). Sentence accuracy refers to the percentage of times where the ASR system recognized the digits in an utterance correctly.

Table 1. ORACLE LEARNING ACCURACIES

Network configuration	Word %	Sentence %
200 hidden nodes (standard, the oracle ANN)	99.59	98.70
OTN (200 100)	99.56	98.60
100 hidden nodes (standard)	99.41	98.10

As seen above, an OTN (200 100), having half as many hidden nodes than its oracle, achieves a comparable accuracy, 99.56% instead of 99.59%. The OTN (200 100)'s accuracy also demonstrates 25% less error than training a 100 hidden node net with the standard back-propagation approach (99.56% vs. 99.41%).

One reason for the improvement is that the OTN can train as long as necessary to over-fit on the oracle ANN's outputs using the large amount of unlabeled data and hence "sees" far more data points than the standard trained network which can only be trained with labeled data. Also the fact that the OTN (200 100) is learning a simpler function than

the 0-1 encoding the standard 100-node network must learn may aid its improved accuracy.

5 Conclusion and Future Work

The results of the experiment support the theory that training a smaller ANN to approximate a larger ANN results in 1) a less complex network capable of accuracy comparable to its oracle, and 2) improved accuracy over standard training of smaller ANNs. An OTN with half the complexity of its oracle had significantly less error than the standard trained model, and achieved comparable accuracy to its oracle.

Future work in this area includes several more experiments. First, research will be done to determine how well even smaller ANNs perform when approximating both the original oracle and even approximating larger OTNs. It is important to determine the relation between the sizes of both the OTN and its oracle ANN. For example, does a 50 hidden node network yield better results approximating the original 200 hidden node oracle or an OTN (200 100)? Next, even more powerful oracles will be obtained (including mixture models, ensembles, etc.) to ascertain the robustness of using OTNs when presented with non-ANN oracles.

Preliminary results in the above areas indicate that the closer the complexity of the oracle ANN to the OTN, the better the OTN performs. For example, in one experiment, an OTN (100 50) achieved higher accuracies than an OTN (200 50). If this trend persists,

the ideal size will be determined (number of hidden nodes) for an OTN to approximate even more complex oracles (mixture models, ensembles, etc.) to reveal how the complexity of an ANN relates to the complexity of non-ANN models.

Other research includes using the above complexity measures to develop a system for more accurately comparing complexity between different classifier models (i.e. ANN compared to mixture-of-gaussian ASR models). The system would be in terms of the number of hidden nodes needed to effectively approximate a given model and would be obtained by simply oracle-training ANNs of various sizes using the model being measured as the oracle. The main problem in this area would be handling the different inductive biases between the models.

The ASR engine used in the experiment uses a decoder that takes as much advantage of the order of the outputs as it does the single highest output. Therefore, in order to determine if oracle learning can be as effective in problems that do not require or lend themselves to decoding, further experiments will compare and contrast decoded and non-decoded problems to find the correlation.

Acknowledgments

This research was funded in part by a grant from *fonix* Corp.

References

[1] Le Cun, Y., Denker, J.S., and Solla, S.A. (1990). "Optimal brain damage ", in *Advances in Neural Information Processing Systems 2*, D.S. Touretzky, Ed., pp. 598--605. Morgan Kaufmann, San Mateo, CA.

[2] Waibel, A. (1989). "Consonant Recognition by Modular Construction of Large Phonemic Time-Delay Neural Networks" in D. S. Touretzky (ed.), *Advances in Neural Information Processing Systems 1*, Morgan Kaufmann.

[3] Le Cun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., and Jackel, L. (1990). “Handwritten digit recognition with a back-propagation network.” In David S. Touretzky, editor, *Neural Information Processing Systems*, volume 2, pages 396-404. Morgan Kaufmann Publishers, San Mateo, CA.

[4] P. Domingos. (1997). “Knowledge acquisition from examples vis multiple models”, in Proc. of the *Fourteenth International Conference on Machine Learning*, pp. 211-218.

[5] Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Mateo, CA: Morgan Kaufmann.

[6] Zeng, X. and Martinez, T. (2000). "Using a Neural Network to Approximate an Ensemble of Classifiers," *Neural Processing Letters*, vol. 12, pp. 225-237.

[7] Breiman, L. (1996a). "Bagging Predictors", *Machine Learning*, Vol. 24, No. 2, pp. 123-140.

[8] Craven. M. W. & Shavlik, J. W. (1993). "Learning symbolic rules using artificial neural networks," in Proceedings of the *10th International Conference on Machine Learning*, pp. 73-80, Amherst, MA. Kaufmann.

[9] Craven, M. W. & Shavlik, J. W. (1996). "Extracting tree-structured representation from trained networks", in D. S. Touretzky, M. C. Mozer and M. Hasselmo (ed.) *Advances in Neural Information Processing System 8*, pp. 24-30, MIT Press.

[10] Leonard, R.G. & Doddington, G. (1993). TIDIGITS speech corpus, http://morph.lds.upenn.edu/Catalog/LDC93S10.html. Texas Instruments, Inc.