LAZY TRAINING:

INTERACTIVE CLASSIFICATION LEARNING

by

Michael Rimer

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

April 2002

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Michael Rimer

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

_____	_____
Date	Tony R. Martinez, Chair
_____	_____
Date	Michael A. Goodrich
_____	_____
Date	Dan R. Olsen, Jr.

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Michael 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 R. Martinez
Chair, Graduate Committee

Accepted for the Department

_____

David W. Embley
Graduate Coordinator

Accepted for the College

_____

G. Rex Bryce
Associate Dean, College of Physical and Mathematical Sciences

ABSTRACT

LAZY TRAINING:
INTERACTIVE CLASSIFICATION LEARNING

Michael Rimer

Department of Computer Science

Master of Science

Backpropagation, similar to most learning algorithms that can form complex decision surfaces, is prone to overfitting. This work presents the paradigm of Interactive Training (IT), a logical extension to backpropagation training of artificial neural networks that employs interaction among multiple output nodes. IT methods allow output nodes to learn together to form more complex systems while not restraining their individual ability to specialize.

Lazy training, an implementation of IT, is presented here as a novel objective function to be used in learning classification problems. It seeks to directly minimize classification error by backpropagating error only on misclassified samples from outputs that are responsible for the error. Lazy training discourages overfitting and is conducive to higher accuracy in classification problems than optimizing current objective functions, such as

sum-squared-error (SSE) and cross-entropy (CE). Experiments on a large, real world OCR data set have shown lazy training to significantly reduce generalization error over an optimized backpropagation network minimizing SSE or CE from 2.14% and 1.90%, respectively, to 0.89%. Comparable results are achieved over eight 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 lazy training performing 10-fold stratified cross-validation. Analysis indicates that lazy training performs a fundamentally different search of the feature space than backpropagation optimizing SSE or CE and produces radically different solutions. These results are supported by the theoretical and conceptual progression from algorithmic to interactive training models.

ACKNOWLEDGEMENTS

Thanks to Tony Martinez, my advisor, who guided me in the writing of this work and in obtaining funding to help support my family during my education. Also to Mike Goodrich, whose championing of satisficing theory led me to the realization of this work. I also wish to thank many other faculty and associates for their time and enthusiasm in discussing this work with me, although they are too many to name here.

I am grateful to my dear wife Olga for her unwavering support and encouragement. Without her sacrifice and understanding this work would not have been possible.

Above all, I am grateful to my Heavenly Father, who is the source of all truth and light, for his inspiration and grace.

Table of Contents

List of Figures

List of Tables

"The discovery consists of seeing what everyone else has seen
and thinking what no one else has thought"

Albert Szent-Georgyi

Chapter 1

Introduction

Artificial neural networks have received substantial attention as robust learning models for tasks including classification and function approximation [Rum85]. Learning is no longer formulated as simply function approximation [Bia98]. Over the last decade much research has gone into improving a model's ability to generalize beyond sampled data. Many factors play a role in a network's ability to learn, including network properties, the learning algorithm, and the nature of the problem being learned. Often, overfitting the training data is detrimental to *generalization* (correctly predicting future unseen data from presently available data). Developing a universal learning model for high-accuracy learning while avoiding perceptible overfitting over relevant (real world) problem domains remains elusive.

This work proposes *interactive training* (IT), an emerging neural network learning paradigm wherein a network or networks learn multiple tasks interactively. Interactive training provides a framework for improving generalization on complex real world classification problems, such as speech and character recognition. Enhancing neural network learning models through interaction is an extension consistent with the evolving

paradigm shift in computer science from reliance on algorithms to interactive models as the driving force in problem solving [Weg97, Weg99].

This work presents interactive classification training, or *lazy training*, as its main contribution, a training technique that implements a new objective function for learning classification tasks. Lazy training seeks to directly minimize classification error by backpropagating error only on misclassified samples from outputs that are responsible for the error. It is able to achieve this by enabling the output nodes to interact among themselves. This technique discourages overfitting and is conducive to higher accuracy in classification problems than optimizing current objective functions, such as sum-squared-error (SSE) and cross-entropy (CE).

Lazy training is shown to perform markedly better on a large OCR data set than an optimized backpropagation network minimizing SSE or CE, reducing classification error from 2.14% and 1.90%, respectively, to 0.89%, a 58.4% decrease in error. 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 lazy training performing 10-fold stratified cross-validation. Analysis indicates that lazy training performs a fundamentally different search of the feature space than backpropagation optimizing SSE or CE and produces radically different solutions.

1.1 Philosophy of artificial neural networks and terminology

A common purpose of feed-forward backpropagation neural networks is to estimate a function $f : \boldsymbol{x} \rightarrow \boldsymbol{y}$, from input variables $\boldsymbol{x} = \{x_1, \ldots, x_m\}$ to output variables $\boldsymbol{y} = \{y_1, \ldots, y_n\}$. Neural networks consist of a connected directed graph of nodes (grouped into layers) and edges. Pattern features ($\boldsymbol{x}$) are entered into the network through the layer of *input nodes*. Results ($\boldsymbol{y}$) are received through the layer of *output nodes*. Optional layers of *hidden nodes* within the network assist in estimating the generating function. Layers of nodes are linked together by connecting edges called *weights* that represent excitatory (positive values) or inhibitory (negative values) signals between nodes. Neural network backpropagation learning is commonly accomplished by repeatedly presenting data samples to the network. The weight values are updated by minimizing an error or *objective* function until the network reaches an acceptable stopping criterion. When the stopping criterion is met the network is said to have *converged*. The process of presenting samples to a network to estimate the true data distribution is known as *training*. The samples presented to the network are known as the *training data*. Future data that are input to the network for processing once training is completed are known as *test data*.

1.2 Overview

An overview of the issues of learning with feed-forward backpropagation neural networks and is presented in Section 2. The problem of overfitting is discussed and existing solutions are summarized. A discussion of objective functions is provided in Section 3. Interactive training is discussed in Section 4. The lazy training heuristic is

presented in Section 5. Experiments and analysis are given in Section 6. The usefulness of interactive processes, problems in training networks independently and the benefits of interactive training are presented in Section 7. Conclusions and future work are presented in Section 8.

Chapter 2

Considerations in Neural Network Training

In multi-layer perceptron (MLP) neural network learning, network speed, complexity and size are important considerations. However, as computing resources continue to increase the consideration of *generalization* stands out at the forefront. This, after all, is the prime purpose of learning. With sufficient capacity, a network is able to store all of the training patterns presented it, and can reproduce results exactly as if performing a "table lookup." After a certain point in backpropagation learning, however, reducing training set error often accompanies an increase in test set error, illustrating the degradation in generalization that accompanies overfit as training continues (see Figure 1).

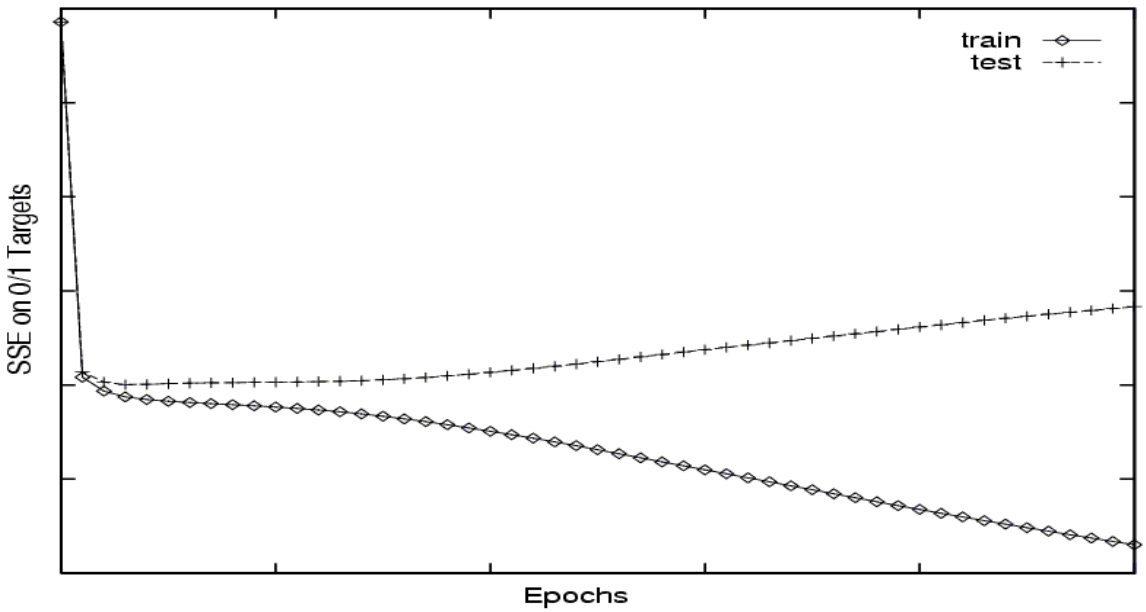

Figure 1. As training error (SSE) decreases, overfit is perceived as test error increases.

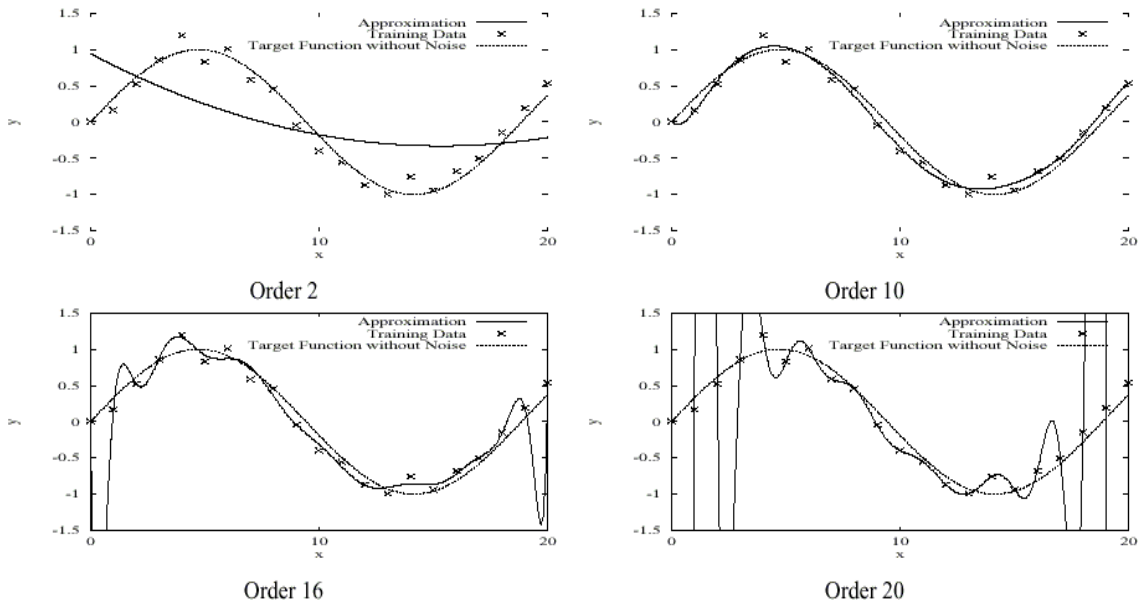

Figure 2. Polynomial approximations of data from y = sin (x/3). Significant overfitting can be observed for orders 16 and 20.

Overfitting has been classically pictured by considering the task of fitting a curve of arbitrary polynomial degree to a set of *N* data points (possibly affected by noise) extracted from a given domain [Law00] (see Figure 2). Often, with a low-order function the data points can be approximated fairly well. With an N^{th}-order polynomial, it is guaranteed that all data points in the sample can be fit exactly. However, this might be a poor solution for other data points in the population. Numerous experiments in the literature provide examples of networks that achieve little error on the training set but fail to achieve "optimal" accuracy on test data [And95, Sch93]. There is an inherent tradeoff between fitting the (limited) data sample perfectly and generalizing accurately on the entire population (see Section 2.6).

The question of how to prevent overfitting is a subtle one. When a network has many free parameters, not only can learning be fast, but also local minima can often be avoided.

On the other hand, networks with few free parameters tend to exhibit better generalization performance [Cas97]. Determining the appropriate size network remains an open problem. How to combat overfitting has received much attention in the literature and is the main focus of this work.

In this section, several issues are enumerated which 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 the lazy training philosophy presented in this work fits into these issues is discussed.

2.1 Altering Network Topology

Network topology plays a large role in achieving high generalization. Most commonly, solutions involve training a *fully connected* network (every node in one layer is connected to every node in the next layer). However, it has often been shown that partially-connected networks perform as well or better than fully-connected ones. Pruning algorithms, such as Optimal Brain Damage [Sol90] and Optimal Brain Surgeon [Sto93], reduce the connectivity in an overly specified network, and construction algorithms (several are enumerated in [And01b]) insert needed connections into a skeleton-network until sufficient function specification is achieved.

2.1.1 Pruning Algorithms

For a fixed amount of data, networks with too many weights often do not generalize well. On the other hand, networks with too few weights will not have enough power to

represent complex data accurately. The best generalization is often obtained by trading off the training error and the network complexity [Lec90]. 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 [Wan94] as not only the number of parameters but also 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 several small weights. Hence, complexity can be reduced not only through pruning parameters, but also by shrinking their values. A learning algorithm that aims at preserving small weights during training can aid in improving generalization. One example of this is performing *weight decay* [Wer88], 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. The lazy training algorithm presented in this work actively attempts to find good solutions with weights remaining as small as possible.

2.1.2 Growth Algorithms

Dynamic network construction algorithms typically start from a very simple basis and progressively add complexity until the training data are acceptably learned. Theoretically, a network can always be grown until it has perfectly learned the training

data. However, at this point it often acts as a table lookup and exhibits poor generalization. Therefore in growing networks it is essential to choose a proper stopping point. Growth and pruning algorithms can be used in conjunction to first grow a network that empirically has the capability to learn the training data, and then prune nodes until accuracy on a holdout set begins to decrease.

Several growth methods append nodes after the current output node, which is disadvantageous since the original output node was not trained specifically as a feature detector for use in the new network. Cascade Correlation [Fah90] and DMP3 [And01b] add nodes before the output node. Lazy training effectively teaches each output node to function as a feature detector that performs in conjunction with the other output nodes, rather than on its own. In this way, their mutual results can be combined into meaningful, non-redundant output.

2.2 Early Stopping

Early stopping strategies [Wan94] 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 samples. In order to avoid this, early stopping strategies try to determine when the problem has been learned sufficiently well, but not yet overfit [And01b].

[Wan94] 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 [And01b].

Lazy training performs an "online" form of early stopping. Rather than stopping training completely when it is detected that the training set is being overfit, lazy training temporarily omits training on *individual patterns* when it is determined that such patterns are being overfit.

2.3 Network Size

It is often believed that networks with too many degrees of freedom generalize poorly. This line of reasoning is based on the fact that a sufficiently large network is able to "memorize" the training data, essentially performing a table lookup. By reducing the learning capacity of such a network, it will be thereby forced to generalize, as it no longer has the capability to memorize the training data.

Caruana [Car97] points out that in order to perform a proper theoretical analysis of net 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 (usually small, random weights) and make the hypothesis more complex (increase 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 across a satisfactory solution, they begin to perform less like small nets, and more like mid-size networks, and so on. If a large net 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 net capacity, and that using a net that is too small will hurt generalization more than using nets that are too large [Car97].

The ability to perform online stopping, which can be combined with standard early stopping techniques, enables lazy training to be more robust in its management of excessively large networks.

2.4 Parameter settings

Every learning algorithm involves the tuning of parameter settings to generate an acceptable solution. Ubiquitous parameters are network topology (number of nodes, number of hidden layers, connectivity), input representation, selection of objective function, and stopping criteria. For backpropagation networks, a learning rate is included. A separate learning rate can be selected for each weight in the network. Choosing a proper learning rate is important, as there exists a finite range of values for

which convergence is possible. Adding a momentum term can help the parameter range of convergence to be nearly doubled [Qia99]. Additionally, countless variants exist where additional parameters are added to vary learning rate and momentum over time, for flat spot elimination by artificially increasing the output sigmoid derivative [Fah88], capping the error backpropagated to a node, and so forth.

2.5 Ensembles

Learners trained in the same problem domain can be combined into an *ensemble*. Classical methods of designing ensembles involve a two-step process, where the networks are first generated independently, and then combined [Sha96]. Combining networks can be done in many ways. A gating network can determine which network samples are sent through for classifying [Cha97a]. Alternately, a sample can be fed into all networks, and their outputs combined through voting [Bau99]. Their answers can also be fed as inputs into another learner that makes the final decision (a process called *stacking*) [Wol92].

Background knowledge can be taken advantage of in certain cases so networks in an ensemble can learn to master specific tasks. A phoneme recognition system is presented in [Ham89] where division of subtasks (recognizing phonemes) is known prior to training. Optical character recognition (OCR) is another application where division of subtasks can be determined prior to training. When this is the case, networks can be selected to learn specific sub-goals (recognizing certain phonemes or letters as opposed to attempting to recognize entire words or all possible classes at once, for instance).

2.6 Bias and Variance

A network's *bias* and *variance* [Gem92] 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)* [Sha96, p. 8].

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.

An ensemble with high variance tends to have low correlation of errors since each network arrives at a significantly unique hypothesis. The ideal ensemble is a set of nets that do not show any coincident errors. In other words, each net has good generalization (low bias), and when a network is in error, the error is not shared by other outputs (high variance) and can be corrected through voting. If only one output is in error, it can be overruled by considering the majority of correct outputs of the remaining nets. However, ambiguity results when more than one (i.e., close to a majority) output is in error. Low bias and high variance is desirable in an ensemble, and having sufficiently high variance

can make up for moderately high bias. Variance is commonly introduced into ensembles by varying the data presented to each network. This can be done through data sampling, disjoint training sets, adaptive resampling, providing different data sources, preprocessing, or a combination of these [Sha96].

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 [Fri97]. Lazy training reduces variance among outputs by "over-smoothing" the decision surface. SSE bias is increased, but lazy training is used for classification tasks, not function approximation.

Variance is controlled by the degree of over-smoothing – more smoothing, less variance. Lazy training accomplishes over-smoothing by discouraging patterns from affecting the shape and location of the decision surface more than is required for correct classification. Lazy training is need-based, attempting to reduce output error only when other outputs reproduce that error to an extent that can be considered detrimental. Lazy training does not attempt to reduce error when it can be resolved by jointly considering the output values (as in ensembles), so as not to increase variance needlessly though overfitting.

2.7 Chapter Summary

Over the last fifteen years, much effort has been put into developing optimized neural network learning models and techniques. Techniques, such as Quickprop [Fah88] and RPROP [Rie93], seek to speed up learning by dynamically adjusting update parameters.

Models that seek to generate network topologies that are more suited to learning a given problem are classified as adaptive learning algorithms [And95, And96, Fah90]. Partially connected static architectures are also considered in [Cha97]. These networks have fewer parameters and are therefore simpler and more efficient than fully connected networks yet are able to perform equally well.

In taking all of the above issues into account, overfitting is typically considered to be a global phenomenon. However, the degree of overfitting can vary significantly throughout the input space. Lawrence and Giles [Law00] 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 [Fri97]. Lazy training seeks to achieve minimal overfitting not only globally but also locally by not training on patterns that are already correctly classified.

The problem of overfitting has likewise received much attention in the literature. Methods of addressing this problem include using a holdout set to stop training early [Wan94], cross-validation [And99a], node growth [Fah90, And01b] and node and weight pruning [Cas93, Cas97], weight decay [Wer88], and ensemble techniques [Sha96], among others. These techniques approach optimal solutions given the bias of standard backpropagation learning but do not consider possible improvements to the bias itself. Node pruning seeks to improve accuracy by reducing network size, rather than alleviating

the problems common to larger networks, for example. This bias exists to a large extent as a result of the objective function used to update the network parameters. A discussion of objective functions is presented in the next chapter.

Chapter 3

Classical Objective Functions

3.1 Critique of common training techniques

Much research has gone into developing optimizations to the backpropagation algorithm. Artificial training sets are often devised to illustrate the behavior of new training models in theoretical situations. However, as neural networks are increasingly used in real world machine learning solutions, the necessity of basing research results on real world applications cannot be ignored. As early as the development of the backpropagation algorithm, Rumelhart touched upon the importance of training and generalizing on real world data sets [Rum85]. However, the validity of results is often still based on analysis of artificial data sets. Tollenaere [Tol90] observes once again it would be better to publish results on real world problems. Now, routine experimentation on real world problems to test the validity of new learning approaches has become more common.

There has been a tendency to base "better" results of novel training variants on measurements which are not conclusive of improved learning ability. This is especially prevalent in earlier research (see [Tol90, Section 3.1] and [Fah88, Section 2.1] for a good discussion of this), but still occurs today. Results in the literature on speed enhancements, for instance, often show how fast a new algorithm can converge to

arbitrary accuracy on a training set. In essence, this demonstrates how easily new training algorithms can *overfit*, but says nothing about their ability to generalize. Further, when test set results on classification problems are reported, they as often as not report the minimization of some objective function used to learn, rather than classification accuracy, which is the real goal of the learner.

3.2 Overview of Objective Functions

Backpropagation neural networks are commonly trained through gradient descent procedures. Gradient descent does not allow direct minimization of the number of misclassified patterns [Dud99, Chapter 5]. Therefore, an objective function must be derived that will result in increased classification accuracy as objective error is minimized. The methodology of gradient descent requires a differentiable output function (*i.e.*, squashing function) in order to minimize objective error. When a sample is presented to the network, the network produces an output value. This output may have a corresponding error measure derived by its deviance from a target output value. 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.

Classification of N classes is often viewed as a regression problem with an N-valued response, with a value of 1 in the nth position if the observation falls in class n and 0 otherwise [Leb93]. 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 htan). Using hard targets is a naïve way of training and does not generalize well. Different fractions of the data are learned at different times during training, and using hard targets not only leads to *weight saturation*, making it harder and slower to learn samples that have yet to be learned, but also forces the learner to overfit on samples that have already been learned. Using hard targets of 0.1 and 0.9 presents a less severe solution.

Rankprop [Car95] provides an alternative method to training with hard target values and Caruana 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.

Below is a critique of several of the most commonly used objective functions, followed by the introduction of the classification training objective function.

3.2.1 Sum-squared error / Mean-squared error

Sum-squared error (SSE) or *mean-squared error* (MSE), a common statistical measure of optimality, is a natural choice for an objective function, as it is differentiable. The most common way to train a neural network is thus to approach network weights that minimize 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). This suggests that other metrics are more suited to classification problems.

In [Lec90], 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 indicates that MSE is not the most reliable objective function for this or similar 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.

3.2.2 Cross-entropy

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 [Mit97] and is therefore more appropriate to classification problems. However, error values using SSE and cross-entropy have been shown [Ham90] to be inconsistent with ultimate sample classification accuracy. That is, minimizing CE as well as SSE is not necessarily correlated to high recognition rates.

3.2.3 Classification figure-of-merit

The *classification figure-of-merit* (CFM) objective function was introduced in [Ham90] for classification problems. It provides a closer estimation of true classification accuracy, as minimizing error is monotonic with increasing classification accuracy. Networks that use the CFM as their criterion function are introduced in [Ham90] and further considered in [Barn91].

However, CFM does not determine localization in learning [Jac91, p.2], *i.e.*, every net is trained on all samples. It produces strong (cooperative) coupling between experts, but there could be redundant experts for each sample. [Jac91] introduces competitive experts. A gating network makes a stochastic decision about which expert to select based on the input. This kind of system tends to devote a single expert to each training case.

The task of CFM is to separate network output values by as large a range as possible. Like SSE and CE, this tactic encourages weight saturation, which is often indicative of overfitting and detrimental to accuracy [Bart98].

3.2.4 Information gain

Information gain is especially useful with iterative network growth in mind [And01b]. Since it make decisions which have the most affect on accuracy, it has the potential to avoid local minima early on in the training process. Rather than each learner trying to immediately fit local regions of a function (curve) to arbitrary accuracy, as one learner fits a small region sufficiently well, others stop trying to learn that part and can direct their attention to other areas still in need of learning the most.

A problem with information gain is that it does not clearly suggest how training is to proceed. Training is carried out either on repeated runs of random weight perturbations or by normalizing the learning rate for each class based on the prospective information to be gained from correcting an error on an incorrectly classified sample of that class [And01b]. Therefore, usually only very simple models are trained to maximize information gain, such as perceptrons.

3.3 Improving the objective function for classification

Methods for overcoming problems in the backpropagation objective function often involve forming network ensembles. Ensemble techniques, such as *bagging* and *boosting*

[Mac97], or *wagging* [And99b], are more robust than single networks when the errors among the networks are not positively correlated (see Section 4).

There is evidence [Bart98] that the size of the weights in a network plays a more important role to generalization than the number of nodes. A simpler method of preventing overfitting is to modify the objective function by providing a maximum error tolerance threshold, d_{max}, which is the smallest absolute output error to be back propagated. In other words, no weight update occurs for a given d_{max}, target value, t_j, and network output, o_j, 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 [Sch93].

3.4 On the appropriateness of the objective function

It must be stressed that minimizing an objective function is not the goal of learning. Rather, it must be viewed as the mechanism that guides the network in learning the concept. The above objective functions provide mechanisms that do not directly reflect the true goal of classification learning, which is to achieve high recognition rates on unseen data. A designer implementing a network to learn the task of face or voice recognition by minimizing SSE is not really interested in what value the SSE reaches, but how accurate the recognizer is. Additionally, inappropriate objective functions can be deceiving in portraying how well the network has learned the problem (e.g., LeCun's *digits* study mentioned in Section 3.2.1). This being the case, the objective function

chosen for learning a given task should approximate the true goal of the learner as closely as possible.

3.5 Lazy training, a novel objective function

This thesis presents the classification error, or *lazy training* objective function. Its philosophy is similar to CFM, in that in general it also attempts to increase the range between output activations. However, lazy training differs from CFM in that it widens the range between outputs only when there is classification error (which in actuality translates to a *narrowing* of the gap between outputs, as they are transposed with respect to their classical target values). When a classification error is made, error is backpropagated only from those outputs that are credited with producing the error. This approach allows the network to relax more conservatively into a solution and discourages weight saturation and overfitting. A theoretical basis for lazy training is presented in the next chapter, and the lazy training error function is presented in Chapter 5.

Chapter 4

Interactive Training

It has been proposed [Weg97, Weg99] that learning models that interact with an external environment (*e.g.*, another learner) have a greater theoretical power of expression than non-interactive models. To support this, [Wei99] shows that coupled agents perform much more efficiently than independent agents at complex learning tasks. The paradigm shift from optimized, but isolated, algorithms to interactive models reflects the current evolution in the philosophy of the field of computer science from procedure-oriented to object-oriented languages and single mainframes to networks of personal computers.

Neural network models that learn interactively are proposed to be superior over independent models. A discussion of combining neural networks into ensembles can be found in [Sha96]. Interactive ensembles are presented in [Liu99a, Liu99b], where the networks in the ensemble are trained simultaneously with the inclusion of an additional error term that encourages negative error correlation among the networks. This generally provides some improvement. However, the field of interactive learning among neural nets is largely unexplored. Lazy training is an original contribution to the budding field of interactive neural network learning.

Interactive methods can be performed on separate, specialized single-output networks or a single multi-output network, but discussion in this section will be limited to separate networks for simplicity. The advantages and disadvantages of each method are discussed in Section 6.3.

4.1 Multi-task learning

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 multiple-output network, is described by Caruana [Car93, Car94, Car95, Car97]. Caruana shows how learning multiple tasks in conjunction (e.g., when learning how to automatically drive a vehicle, recognizing where the road's center line and/or left and right lanes are located in addition to deciding just how to steer) helps to avoid local minima and improve generalization. MTL performs better (learning tasks simultaneously) than learning tasks separately [Car93]. There are several reasons why MTL improves on single-task learning (STL). Advantages and disadvantages of using MTL over STL are discussed in [Car93].

A problem that often occurs during training is due to the moving-target problem, and is referred to by Fahlman [Fah91] as the *herd effect*. Suppose there exist two separate computational sub-tasks, A and B, which must be performed by the hidden nodes of a network. Any of the hidden nodes could handle either task, but since they have no way

of communicating among themselves, they must decide independently which task to solve. If task A produces a more coherent error signal than B, there is a tendency for all the hidden nodes to focus on learning A. Once A is redundantly solved by all the nodes in the network, then they begin to work on B, which is now emitting the only significant error signal. If they move toward B all at once, then problem A reappears. Often the weights can be observed to oscillate for a prolonged period of time until the "herd" finally splits up and deals with both sub-tasks at once. This phenomenon, where weights oscillate back and forth, can be seen on complex problems (e.g., *2 spiral*). The herd effect can be observed not only among nodes in an MTL or STL network, but among networks being trained simultaneously in an ensemble as well.

4.2 Interactive learning

Interactive training (IT) goes a step beyond MTL. In addition to simply having specialized networks learning tasks at the same time, IT explicitly shares relevant information among the networks during training to coordinate their learning process. When networks are trained concurrently, rather than sequentially as in standard ensembles, they can take advantage of greater expressive power through interaction during the training process. Two or more networks can collaborate together to decide how learning is to proceed at any given point. One method for allowing interaction by simultaneously training ensemble networks is presented in [Liu99a, Liu99b].

Lazy training is another instance of the interactive training philosophy. Network interaction through lazy training allows the implementation of a *dynamic error threshold*.

That is, when one network presents a sufficient solution in an area of the problem space, other networks do not need to work at redundantly modeling the same local data. Consequently, they are able to specialize and break up a complex problem 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 algorithm is presented in the following chapter.

Chapter 5

The Lazy Training Heuristic

Interactive training considers the output activations of the entire system during training. There are several possible ways to utilize this information in calculating the error signal for backpropagation. In this case, for each pattern considered, only those networks that are credited with a classification error are updated through backpropagation. The result is training without idealized target outputs of 0 and 1, providing a training mechanism that is reminiscent of constraint satisfaction and reinforcement learning, where the network learns to interact with its (changing) environment. As this technique forces networks to learn only when explicit evidence is presented that their state is a detriment to classification accuracy, we have dubbed this technique classification training, training by necessity, or lazy training (not to be confused with *lazy learning* approaches [Aha97]).

Lazy training is similar to Rankprop in that it avoids the use of hard target values. However, rather than providing "soft" targets, it avoids the use of specific target values all together. The objective of lazy 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. When target values are not required to guide training, the network is able to arrive at a solution more simply. This avoids many of the problems

accompanying overfitting.

Network weights are updated during lazy training exclusively to minimize classification error. When the network misclassifies a sample, credit for the error is assigned to two sources. The first is the set of output nodes with higher output values than the target output node (resulting in the system outputting the wrong class value). The second is the target output node itself, which outputted too low a value to produce the correct classification. By lazy training, smaller weights, even approaching zero, can provide an acceptable solution for classification tasks. In an MLP, weights are updated based on how much each node contributes to the output error. Analogously, the IT model updates only the weights of nets that contribute to hindering the classification accuracy of the system. Incorporation of an error function minimizing the classification error directly is made possible through the increased interactive expressiveness of the IT network. This approach is formalized as follows.

5.1 Lazy Training error function

Let N be the number of output nodes in a network. Let o designate the output value of a node ($0 \leq o \leq 1$ for sigmoid). Let o_k be the output value of the k^{th} output node in the network ($1 \leq k \leq N$). 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 in the general case for more

than one output node to act as a target node for a given class label. 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*, or formally

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

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 [Joo98].

To illustrate how lazy training works, consider a three-class problem with a target vector of (0, 0, 1) for a given pattern. On this pattern, the 3-output network outputs (0.1, 0.2, 0.4). While the third output (the target) has significant 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).

Only if one of the competitors fire higher 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 outputs would backpropagate error: the second since it fires higher than the target node, and the third, since a non-target node fired higher than it.

5.2 Advantages of lazy training

Repeatedly forcing output values closer to 0 or 1 in cases where pattern classification is already correct usually results in overfitting. This needlessly increases network variance (sensitivity to the training data), increasing classification error on the test set. This insight is the driving philosophy behind lazy training, which avoids this practice.

Lazy training a network proceeds at a much different pace than using 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 significantly fewer weight updates overall (proportional to the classification accuracy) and allows the model to relax more gradually into a solution (picture an airplane gliding down in the direction of the global error minimum as opposed to a ball rolling along the steepest error gradient). As a result it encounters fewer local minima. Lazy 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 given magnitude, lazy training dynamically halts learning at the *first possible point* that correctly classifies a training pattern. This can be considered using a *dynamic error threshold* that is unique for each training pattern and network state. The result is training without idealized target outputs, or target values at all for that matter, since a sample can be potentially "learned" with any output, providing competitors output lower values.

5.3 Increasing the margin when lazy training

Overfitting is minimized through lazy training in another regard because outliers (noisy samples) 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. This translates to halting the movement of the decision surface right next to the sample (see Figure 3b). This is in contrast to classical SSE training, where

hard target values of 0 and 1 require pushing the decision surface as far away from all points as possible, even noisy outliers (see Figure 3a). Hence, a test pattern (represented by the question mark) falling immediately next to a noisy outlier belonging to a competing class has a much better chance of being correctly classified. In other words, network variance is substantially reduced.

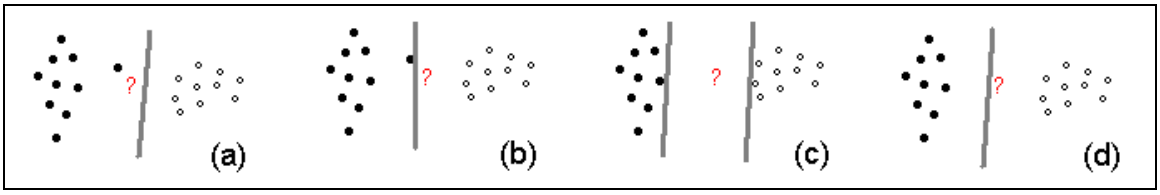

Figure 3. SSE decision surfaces (a,d) and lazy-trained decision surfaces (b,c,d).

When lazy 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-firing node (see Figure 4). This is true for correctly classified samples (those above 0 in Figure 4), and also for misclassified ones (those below 0). This means that most training samples 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 3c. According to the application this might not be desirable.

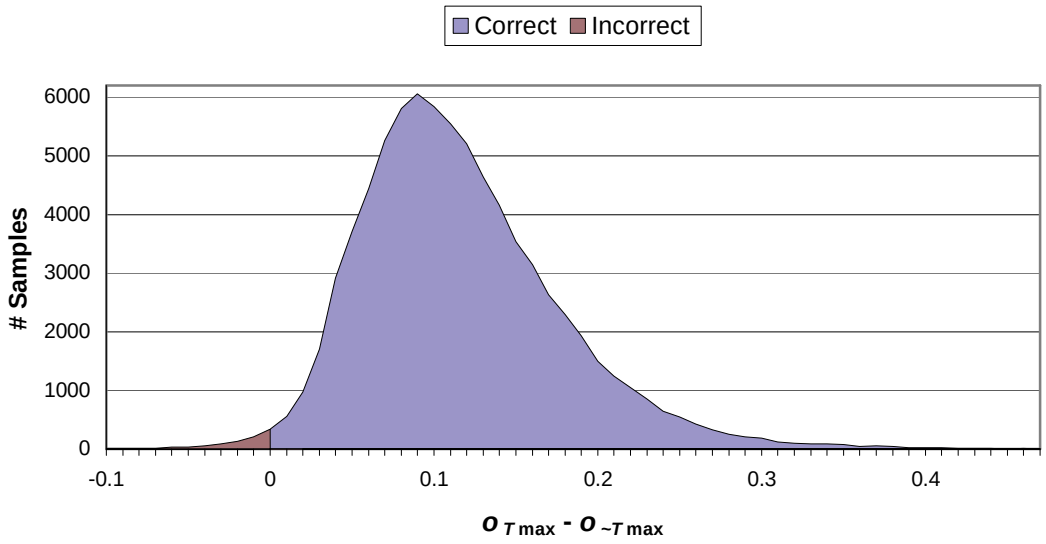

Figure 4. Network output error margin after lazy training.

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 modification to (1) is presented as

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

In this way, lazy training approximates the SSE solution and the margin is maximized even in the absence of outliers (see Figure 3d).

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. Analysis of updating μ during training will be performed in future work.

Including a margin also decreases the amount of "classification oscillation" that occurs as outputs react to one another. Since patterns will lie close to the decision surface, many frequently slide back onto the wrong side as the decision surface shifts around while training proceeds. Requiring them to be situated further away from the dividing hyperplane reduces the incidence of renewed misclassification, leading to quicker convergence.

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

Chapter 6

Experiments and Analysis

Neural networks were trained through backpropagation minimizing SSE, CE and through lazy training to explore the advantages of interactive training techniques over independent learning. These models include:

- Single-output networks on two-class problems (positive samples are assigned a target value of 1.0, negative samples are assigned a target value of 0.0)
- Multi-output networks (one output per class)
- Independent single-output networks on multi-class problems (one per class)

Various styles of network interaction were considered, including:

- Among *N* networks for *N*-class problems (one per class)
- Between dual (complementary) networks for each class (two per class)
- Within a single lazy-trained multi-output network (one output 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)

Real world problems were drawn from the UC Irvine Machine Learning Database Repository (UCI MLDR) [Bla98] and from a large database of machine printed characters gathered for OCR. This provides a vantage point from which to evaluate the robustness of the lazy training heuristic.

In empirical comparisons among different learning methods, appropriate training parameters were determined in order for each model to maximize generalization. For further conceptual analysis and illustration of the behavior of interactive versus non-interactive systems, results of experiments using a range of parameters are provided.

6.1 Data sets

The performance of independent versus interactive lazy training models has been evaluated on an OCR data corpus (*OCR*) consisting of over 495,000 alphanumeric character samples, partitioned into roughly 415,000 training samples and 80,000 test samples. This work was first presented in [Rim00a].

Two network topologies were evaluated for learning *OCR*, a single multi-output network and *N* single-output nets.

Additionally, eight well-known classification problems were selected from the UCI MLDR. 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. 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 is used to learn these problems. Results on UCI MLDR problems were gathered using 10-fold stratified cross-validation.

6.2 Training parameters

Experiments were performed comparing the SSE and CE objective functions against lazy training. Fully connected feed-forward MLPs 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}) 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 sample shuffling was used across all networks tested.

On UCI MLDR data sets, training continued until the training set was successfully learned or until training classification error ceased to decrease for 500 epochs. The network model then selected for testing was the one with the best training set classification accuracy. In all *OCR* experiments presented, each MLP in a multiple network model contained a single hidden layer comprised of 32 hidden nodes. The learning rate was 0.2 and momentum was 0.5. Training was halted after 500 epochs.

Network architecture was optimized to maximize generalization for each problem and learning heuristic. Optimized numbers of hidden nodes used for learning UCI MLDR data sets are listed in Table 1. Learning rate was 0.1 and momentum was 0.5 for all UCI MLDR problems.

Table 1. Network architectures on UCI MLDR problems.

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

Data set	**SSE Net**	**CE Net**	**Lazy Net**
ann	21-30-2	21-30-2	21-30-2
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

Pattern classification was determined by *winner-take-all* (the class of the highest outputting node is chosen) on all models tested. When a single-output network is trained on a two-class problem, outputs of 0.5 or higher are considered as outputting positive and outputs below 0.5 are considered as outputting negative.

6.3 Results

6.3.1 OCR data set

Table 2 displays the results of standard SSE and CE backpropagation versus lazy training on *OCR*. *Train %* and *Test %* are the final training and test set accuracy in percent. *Train MSE* and *Test MSE* are the mean squared errors for the training and test sets on the epoch for which the highest test set accuracy is achieved.

Table 2. Results on *OCR* data set.

Method	**Train %**	**Train MSE**	**Test %**	**Test MSE**
SSE (Multiple nets)	99.28	**.0047**	97.86	**.0092**
SSE (Single net)	98.40	.0225	98.38	.0335
CE (Multiple nets)	99.37	.0094	98.10	.0110
CE (Single net)	98.62	.0153	98.58	.0300
Lazy Train (Multiple nets) $\mu = 0.05$	**99.61**	.1830	**99.11**	.2410
Lazy Train (Single net) $\mu = 0$	99.15	.1594	98.96	.1800

The results on *OCR* show that multi-task learning (MTL), or using a single network with multiple output nodes, performs better than using a separate network to learn each 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 is improved.

Observe that training set accuracy is largely preserved on the test set when using a single multiple output network 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. When using 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 lazy training.

Optimizing CE on this difficult classification problem trains and generalizes better than SSE, and lazy training performs significantly better than both. Network models generated through lazy training have the capability of improving generalization even more. These tests also show that, although the final SSE for lazy training is 10-20 times greater than for SSE and CE training, the amount of overfitting is sharply reduced and generalization is improved (see Section 6.4.3). Since the networks learn together their errors are less correlated and the solution transfers well to unseen data.

Generalization error with the best lazy training architecture is 45.1% less than the best architecture trained with SSE and 37.3% less that the best architecture trained with CE. Considering only multiple-output networks, error drops from 1.62% for SSE to 1.42% for CE, and to 1.04% for lazy training, error reductions of 35.8% and 26.8%, respectively. Considering only the multiple-network models, error drops from 2.14% with SSE to 1.90% with CE, and to 0.89% with lazy training, error reductions of 58.4% and 53.2%, respectively.

6.3.2 UCI MLDR data sets

Table 3 lists the results of a naïve Bayes classifier (taken from [Zar95]), standard SSE and CE backpropagation, and SSE and CE lazy training on eight UCI MLDR classification problems. Results were gathered using 10-fold stratified cross validation. 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.

Table 3. Results on selected data sets from UCI MLDR using 10-fold cross-validation. Best values are shown in bold and second best in italics.

Data set	**Bayes**	**SSE**	**CE**	**Lazy SSE**	**Lazy CE**
ann	**99.7** **0.1**	98.25 0.54	98.33 0.53	97.62 0.47	*98.76* *0.51*
bcw	93.6 3.8	96.96 2.01	97.06 **1.81**	*97.22* 2.01	**97.36** **1.81**
ionosphere	85.5 4.9	89.00 4.72	*90.80* 4.64	90.60 **3.75**	**90.88** *3.87*
iris	94.7 6.9	93.83 5.68	94.37 5.87	**95.47** *5.31*	*95.37* **5.25**
musk2	97.1 0.7	99.06 0.37	98.56 0.62	*99.15* *0.36*	**99.27** **0.29**
pima	72.2 6.9	76.26 *4.24*	76.11 4.36	*76.69* **3.43**	**76.82** 6.46
sonar	73.1 11.3	76.06 9.37	78.87 9.03	*80.77* *9.02*	**81.92** **8.60**
wine	94.4 5.9	96.29 4.45	96.74 4.13	**98.31** **3.49**	*97.19* *3.47*
Average	88.79 5.06	90.69 3.92	91.35 3.87	*91.97* **3.48**	**92.20** *3.79*

Overall, an optimized SSE or CE backpropagation network is superior to a naïve Bayes classifier on the above classification problems. A lazy trained network performs better

than both Bayes and standard backpropagation classifiers except on a single data set, *ann*, where the Bayes classifier performed best of all. Cross-entropy based training performs slightly better than SSE reduction on average for both standard and lazy backpropagation training. Lazy CE training performed the best on five of the eight data sets and better than non-lazy trained classifiers on all eight of the eight. Lazy SSE performed the best on two of the eight data sets and better than non-lazy trained classifiers on six of the eight. This seems to indicate that either lazy error reduction metric is acceptable, with CE being preferred.

The average decrease in classification error is from 9.31% for SSE training to 8.03% for lazy training, a 13.7% decrease in error. An overall decrease in standard deviation also indicates that lazy training is more robust to initial parameter values and pattern variance then standard backprop. This reflects the expectation that overfit is reduced and generalization is improved by lazy training.

[Zar95] includes results and analysis of several other machine learning models on these and many other UCI MLDR data sets. Lazy training significantly outperforms all other learning approaches shown on the problems tested here.

6.4 Discussion

Standard backpropagation and other gradient descent learning techniques do not consider or attempt to minimize the number of correctly classified training patterns (see [Dud99, p. 228, Figure 5.11]). Lazy training incorporates a more direct minimization of

misclassified patterns in gradient descent procedures by reducing error on only misclassified patterns.

It does not modify weights to provide a monotonic decrease in a global error signal based on ideal target values, such as SSE or CE. In fact, during training SSE (MSE) often remains roughly constant as accuracy is improved (see Figure 5). Change in CE displays the same behavior as MSE but is omitted from this graph and the following discussion for clarity. This is in contrast to the steady drop in SSE asymptotically approaching zero exhibited in Figure 1 that is illustrative of standard backpropagation training minimizing SSE.

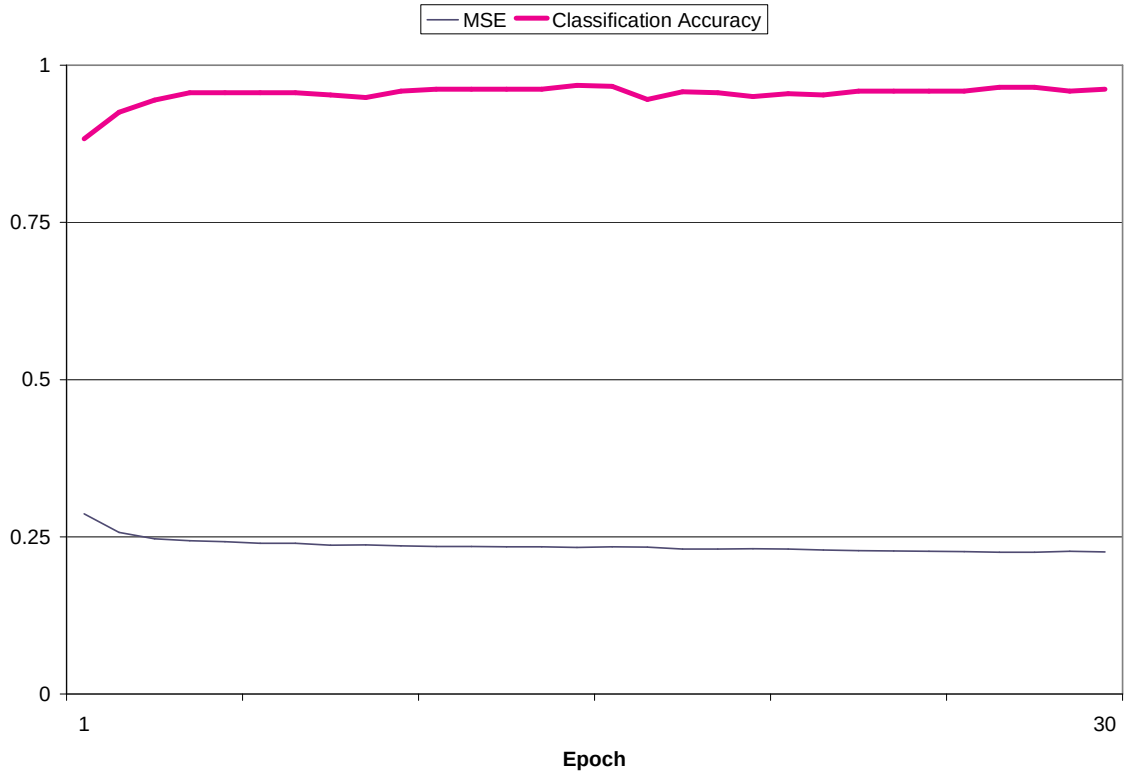

Figure 5. Classification accuracy and MSE during lazy training.

SSE, rather than converging to zero, instead remains rather large. A MSE of almost 0.25 is equivalent to a mean error of nearly 0.5, which illustrates that many output activations are firing about 0.5. This indicates that the weights for these outputs are approximately zero. A large MSE is incurred by pattern outputs being very far away from their idealized target values. This suggests that lazy 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 just as high or higher as when training starts on a network initialized to small random weights.

Figures 6 and 7 give insight into the behavior of the network during the learning process using four objective heuristics. The surface plot shows a histogram of the values output by the network on the training samples every tenth training epoch. Figure 6a and 6b show learning minimizing SSE, and Figure 7a and 7b show behavior during lazy training. The results shown here are only for the *bcw* data set, but such behavior is indicative of all the data sets tested.

Note that SSE training forces the network to output values approaching 0 and 1, even from the very first trace (the tenth epoch). Using a d_{max} of 0.1 reduces this tendency somewhat. Observe the flattened peaks for positive samples in Figure 6b that do not exist in 6a.

Lazy training produces a starkly different behavior. In Figure 7a, it can be observed that all samples output around 0.5 during the entire training process. In Figure 7b, incorporating a confidence margin of $\mu = 0.1$ widens the spread of output values, even causing the outputs of the two classes to visibly split apart as training progresses.

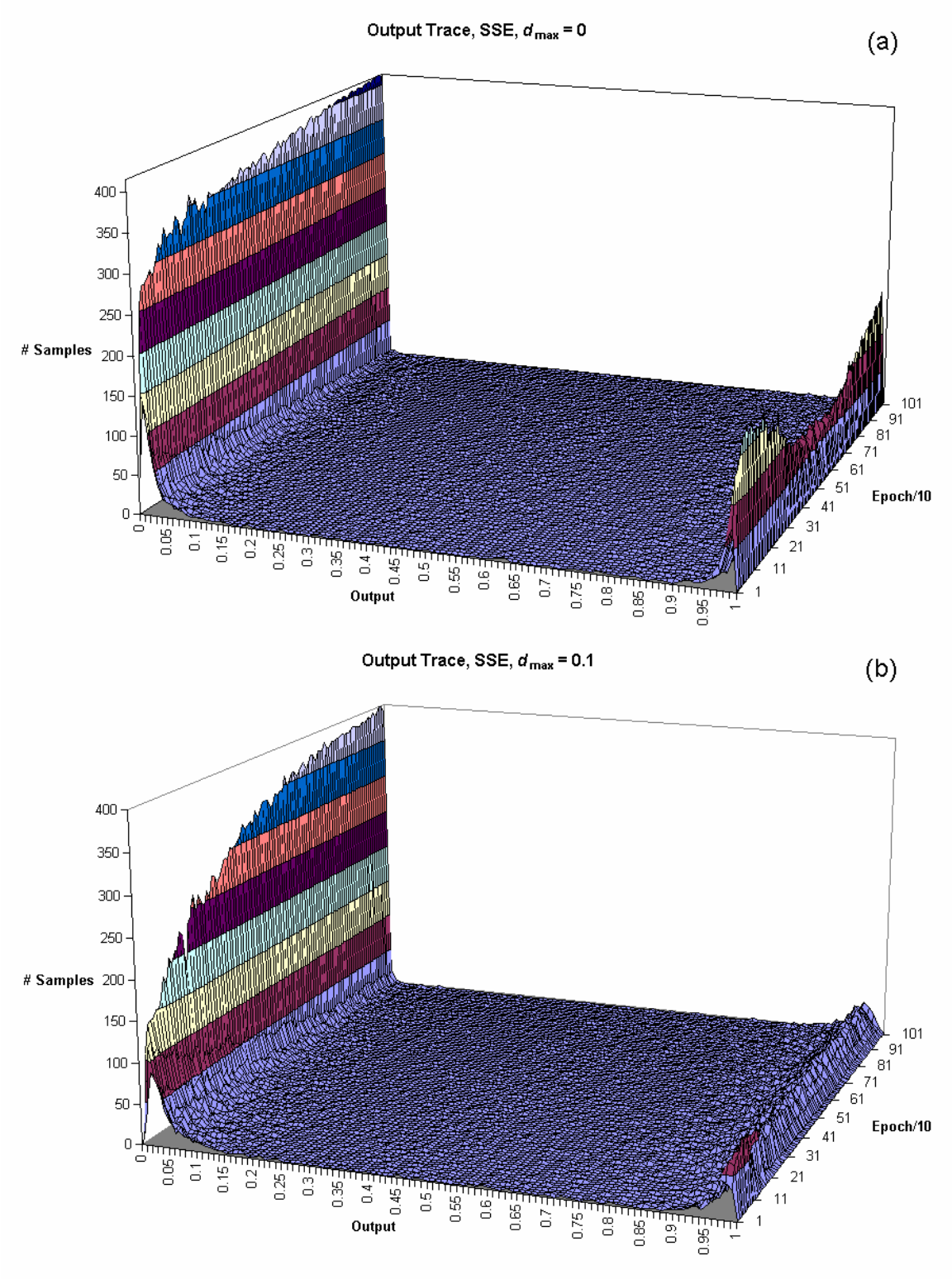

Figure 6. Network output trace during SSE minimization on *bcw*.

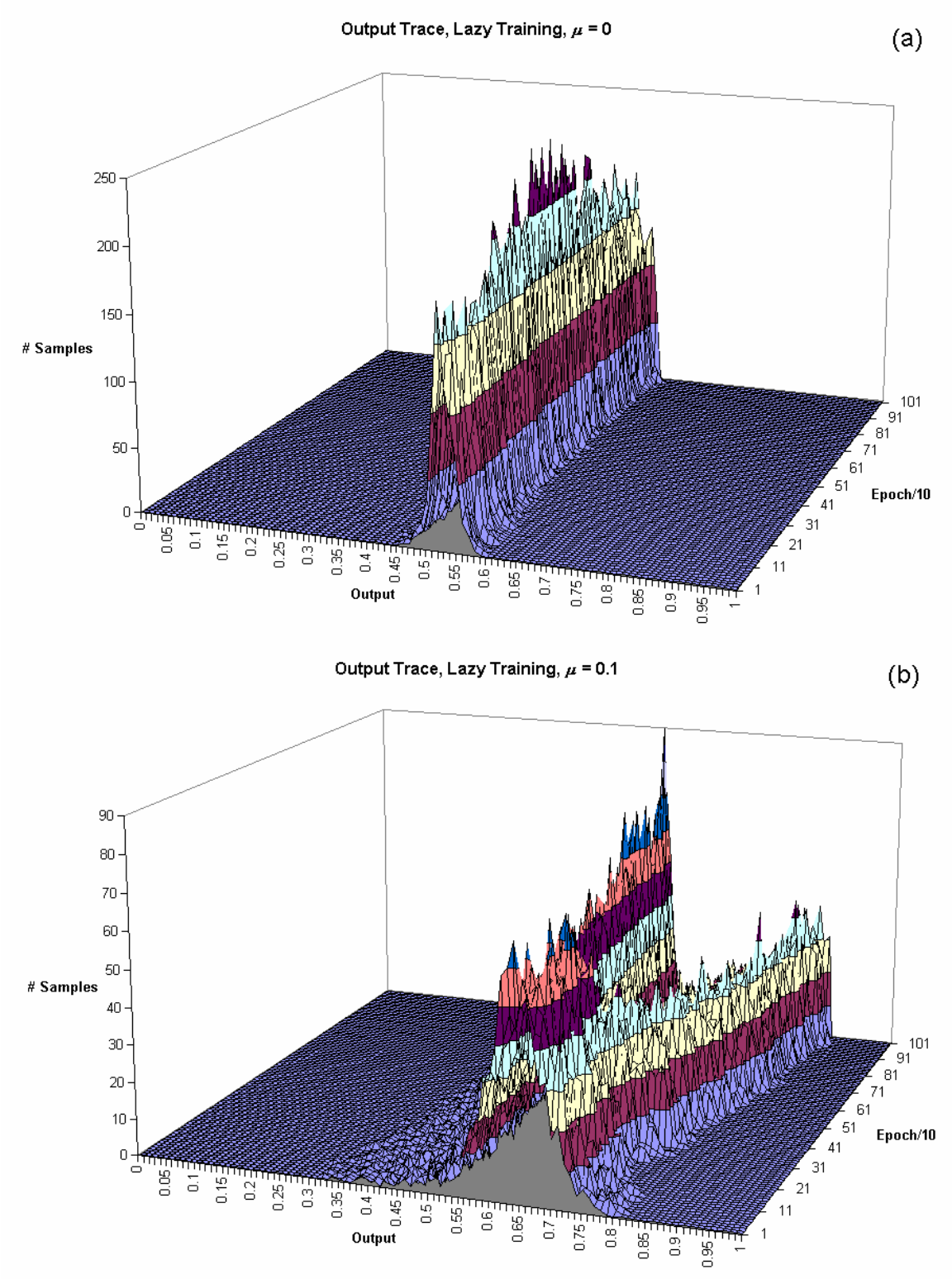

Figure 7. Network output trace during lazy training on *bcw*.

6.4.1 Empirical effects of error margin

Table 4 depicts the results of lazy training on *bcw* with a range of values for μ from 0 to 0.9. The top value is averaged classification accuracy and the bottom value is standard deviation using 10-fold stratified cross validation.

Table 4. 10-fold CV results for lazy training on *bcw* with μ. Best results are in bold.

$\mu = 0.0$	0.01	0.05	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
95.17	**97.22**	**97.22**	**97.22**	96.78	96.93	96.78	95.90	95.61	95.90	95.75	95.75
2.09	2.42	**1.79**	2.01	1.96	2.29	**1.72**	**1.69**	2.94	2.26	2.32	2.40

These results show that lazy training is fairly robust to the selection of μ. A μ greater than 0 causes the decision surface to be further removed from test patterns in general and increases generalization as a result. Values closer to 0 show the most improvement and μ values closer to 1 cause lazy training to revert proportionally to the behavior of standard SSE training. (Note that the accuracy shown for $\mu \sim 1.0$ does not match the accuracy for SSE training in Table 3 because the accuracies in Table 3 are based on roughly optimized parameters for each error function, and lazy and standard training have different optimal learning parameters.)

6.4.2 Effect of SSE on output values

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

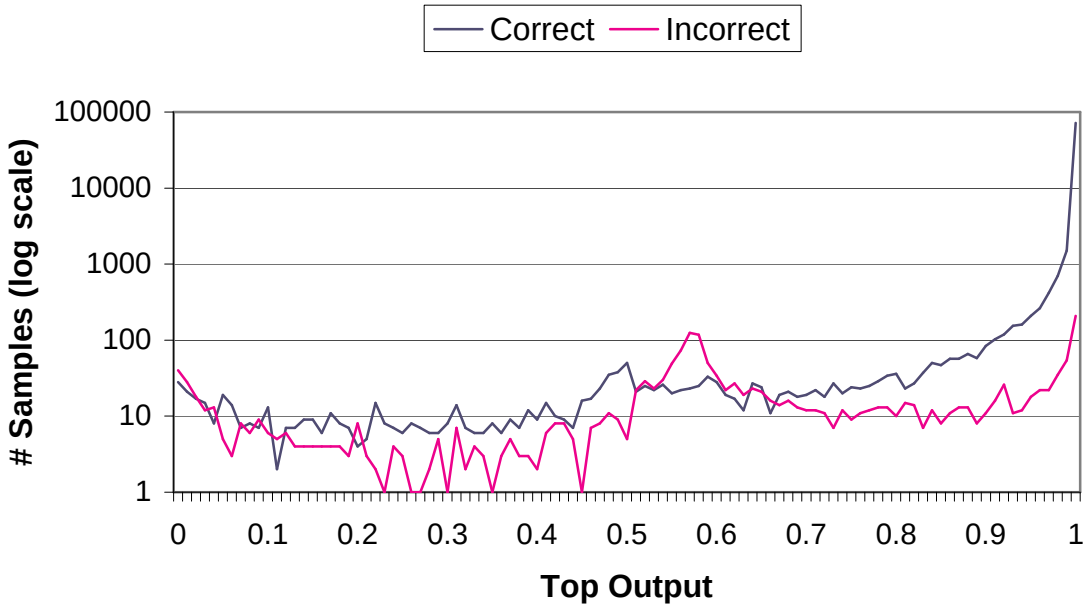

Figure 8. Network outputs on *OCR* test set after SSE minimization.

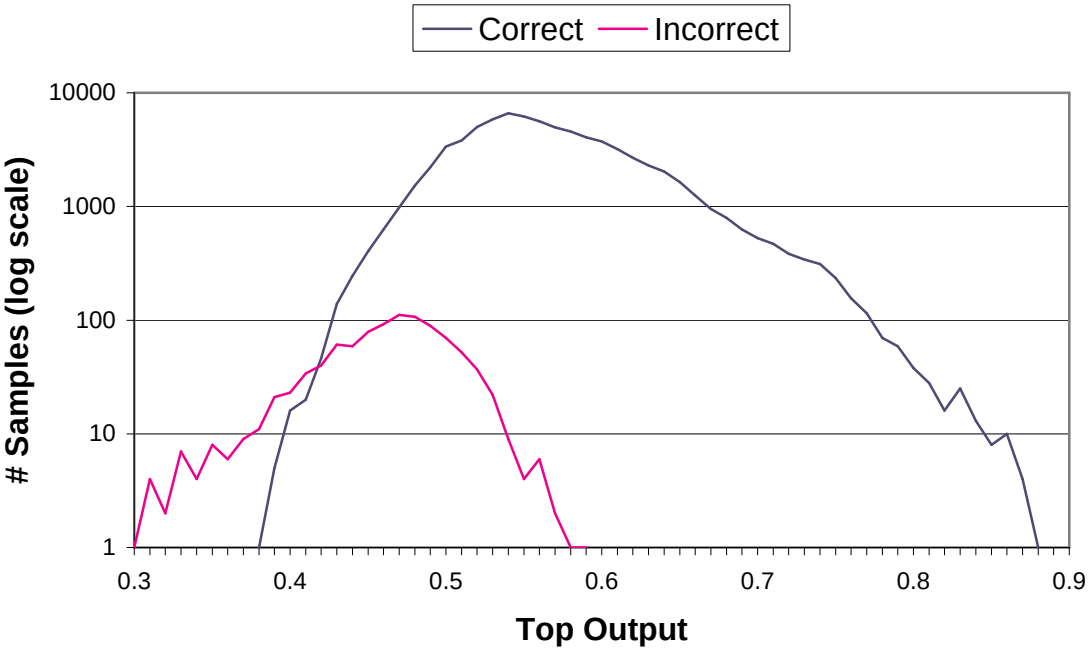

Figure 9. Network outputs on *OCR* test set after lazy training.

6.4.3 Effect of lazy training on output values

Lazy 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 sample 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). Sample 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 standard 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 samples usually have outputs below 0.5 and are lower than the output for correctly classified samples in the majority of cases.

6.4.4 Network complexity

At first, it seems counter-intuitive that networks firing 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, an undesirable trait.

However, it has been illustrated how the number of nodes in a network is not as influential as the *magnitude* of the weights [Bart98]. 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 [Wan94] 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.*

Occam's principle stands, in that it is best to make minimal *use* of the capacity of the network for encoding the information provided by the learning samples [Wan94]. In light of this, it is understandable why training (often overly) complex networks using weight decay or lazy training, which allow networks to converge with smaller weights than normal, perform well. Although the IT network has a high number of parameters, lazy training prevents further weight updates once samples are correctly classified and results in low 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 lazy training,

respectively. The columns denote the average 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 values are bolded. The lazy-trained 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 lazy-trained network is a simpler solution than the networks produced by backpropagation training optimizing SSE or CE.

Table 5. Average final network weights.

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
Lazy	**0.56**	**0.02**	**0.31**	**0.74**

6.4.5 Lazy training single vs. multiple networks

There are advantages to lazy training a single multi-output network over separate single-output networks. 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.

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 much 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 obvious. Not only is memory conserved, but classification speed is increased as well.

Caruana states that 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 hurt 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 [Car94]. However, taking 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.

Lazy 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).

Note, however, that the best *OCR* test accuracy obtained was using multiple networks. It appears their increased computational ability enables them to find a better solution than with a single multi-output network, while lazy training helps them to avoid local minima. Using multiple networks makes more parameters available for use as needed in solving the problem, while lazy training discourages abuse of the increased potential of the system to overfit.

6.4.6 Computational Cost

Lazy training requires an O(*n*) search through the *n* network outputs to determine the highest target and competitor values. This search could be made O(log *n*) if the output values were sorted into a binary search tree as they are produced. However, this additional overhead is negligible compared to the computation requirements of O(*ih*) for feed-forward and O(*ihn*) for backpropagation steps, where *i* is the number of inputs and *h* is the number of hidden nodes. In fact, lazy training saves O(*ihn*) steps by omitting the error backpropagation step when presented samples that it already correctly classifies.

Chapter 7

Philosophy of Interactive Training

Algorithms and Turing machines have been the dominant model of computation during the first 50 years of computer science. Wegner, et al, [Weg99] formalize the claim that interactive finite computing agents are more expressive than Turing machines and consider the evolution from algorithmic to interactive computing models. Their definitions of interaction machines, behavior, and expressiveness are presented here in brief. The reader can refer to their work in [Weg97, Weg99] for a more complete discussion.

Turing machines (TM) are state-transitional string processing machines, of the form $M = (S, T, s_0, F)$, with finite states S, tape symbols T, starting state s_0, and a state-transition relation F: $S \times T \rightarrow S \times O$. They compute functions $f : X \rightarrow Y$ from integers to integers (strings to strings).

Classical Turing machines are not able to interact within distributed systems. *Interaction machines* (IM) provide a stronger model of computation than Turing machines that better capture computational behavior of finite interactive computing agents. *Sequential interaction machines* (SIMs) are state-transitional stream processing machines $M = (S, I,$

m), where *S* is an enumerable set of states, *I* is an enumerable set of inputs, and the transition mapping $m : S\text{x}I \rightarrow S\text{x}O$ maps state-action pairs to new states and outputs. Each step of a SIM can be considered a complete Turing machine computation, and the behavior of SIMs is expressed by I/O streams, of the form (i_1, o_1), (i_2, o_2), …, where o_k is computed from i_k but precedes and can influence i_{k+1}. *Multi-agent interaction machines* (MIMs) can interact simultaneously with multiple autonomous interaction streams.

System behavior of a finite computing machine *M* is specified by its set of all possible interactions with all possible environments, and is denoted by *B*(*M*). *Observable behavior* of a machine *M* for an environment *E* is the set of all behaviors of *M* in *E* and is denoted by $B_E(M)$. Observable behavior is a projection of system behavior to a specific context. Given two finite machines, M_1, M_2, the exclusive-or $B_E(M_1) \oplus B_E(M_2)$ is called the distinguishability set DS for M_1, M_2 in *E*. Two machines are *distinguishable* if DS is non-empty and equivalent if they are not distinguishable.

Expressiveness is the set of behaviors $B_E(M)$, which measures the ability of agents to make observational distinctions about their environment. Machine M_1 is more expressive than machine M_2 in environment *E* if $B_E(M_1)$ is a strictly larger set of behaviors than $B_E(M_2)$. Interaction between an environment and an observer can be modeled as a producer/consumer relation, where *E* is a producer of behavior, and *M* is a consumer. Since the behavior $B_E(M)$ depends on *E* as well as *M*, environments are classified into classes that support progressively stronger forms of interaction:

- *TM environments (relation of enumerable index)*

- *SIM environments (relation of nonenumerable index)*
- *MIM environments*
- *The physical world, W*

Behavior of agents in TM environments is expressed by input-output pairs, while behavior in a SIM environment is expressed by sequences of observations (I/O streams) that allow more finely-grained distinctions among behaviors than single observations. Interaction machines that interact with the real world, *W*, have the property that EQ(SIM,*W*) is a finer equivalence relation on the real world than EQ(TM,*W*). This leads to the following lemma:

$B_W(TMs) \subset B_W(SIMs) \subset B_W(MIMs)$, *where* $\subset$ *is set inclusion for classes of behaviors.*

However, to a TM observer, all computing agents including SIMs and MIMs appear to behave like a TM:

$$B_{TM}(TMs) = B_{TM}(SIMs) = B_{TM}(MIMs)$$

while to a TM, all computing environments as well as the physical world appear to behave like TMs:

$$B_{TM}(TMs) = B_{SIM}(TMs) = B_{MIM}(TMs) = B_W(TMs)$$

Once the idea of expressiveness as distinguishing power is introduced, many levels of expressiveness may be distinguished. Let SIM_k be the class of SIMs restricted to no more than k interactions. For all $k>0$, the distinctions than can be made by SIM_{k+1} in environment W are greater than those by SIM_k. SIMs determine an infinite expression hierarchy with TMs at the bottom, corresponding to behavior $Bw(SIM_1)$:

$$For\ all\ k>0,\ Bw(SIM_k) \subset B_W(SIM_{k+1})$$

and

$$B_W(TM) = Bw(SIM_1).$$

SIMs model interactive question answering. The expressiveness hierarchy for SIMs shows that on-line questioning that makes use of follow-up questioning has greater expressiveness than off-line questioning. Mitchell [Mit97] states the advantages of allowing a learner to query the environment in cases of uncertainty in order to refine its hypothesis.

7.1 Increased expressiveness in neural network training

This expressiveness hierarchy can be related to neural network learning (and other forms of machine learning). The behavior of a neural network during training is that of a Turing machine. It is given training data as input and generates its hypothesis as output:

$B(SIM_1)$: one interaction, that of receiving training data (input string).

Ensemble methods, such as bagging, boosting, and wagging provide a form of interaction. Ensembles often include a two-step training process, that of training each network, and then combining their results once training is finished. Thus, the interaction occurs at a certain point in time, usually when the current network has converged. Ensembles hence provide a greater level of interaction than standard training:

> $B(SIM_2)$: two interactions, training *n* nets (observing training data) and then combining them after training.

This has shown to improve generalization over isolated networks with no interaction [Sha96].

It is conceivable that more frequent interaction will further improve generalization. A logical extension of the "off-line" ensemble process is training the networks simultaneously and enabling each network to observe the behavior of other networks in the ensemble to decide how training should proceed (as in [Liu99a, Liu99b]):

> $B(SIM_t)$: interaction at every pattern during training

Training classifiers together allows them to receive the advantages of increased expressiveness. Dual-network interaction ($N = 2$) produces SIM behavior, while multiple network interaction ($N > 2$) produces MIM behavior.

7.1.1 Expressiveness of perceptrons

Initially, the perceptron model [Ros58] provided a statistical mechanism by which data points existing in a hyperspace of arbitrary dimension could be distinguished by a decision surface, geometrically represented by a hyperplane, comprised of a linear combination of feature values (see Figure 10). The Widrow-Hoff rule [Wid60], and the Perceptron learning algorithm [Ros58], provided techniques by which each feature weight was iteratively refined to provide a more accurate separation of data points from opposing classes. The perceptron is able to distinguish between any two arbitrary data distributions, provided they are linearly separable [Min69].

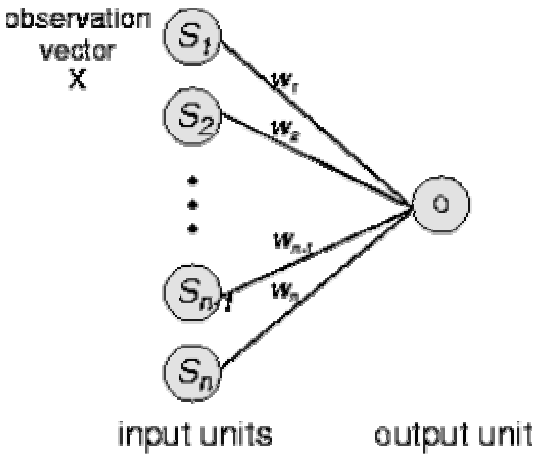

Figure 10. Single-layer perceptron.

7.1.2 Limitations of linear separators

Learning first order features or rules performs well on real-world data, whereas not so well on artificial data. Experts often already have a good intuition of the important features [And99b]. When “higher-order” features have high output correlation, experts

often put them into the problem database as first order features. This explains why perceptron models often perform adequately on real-world data. Still, inasmuch as a solution is non-linear, simple perceptron models are not able to obtain it. As the demand for high-accuracy learning models to solve complex real world problems increases, linear solutions become inadequate.

7.1.3 Increasing expressiveness

Networks with simple architectures generalize better when allowed to interact through ensembles [And99b, Sha96]. Some network dynamic construction algorithms [Fah90, And01b] also allow a form of interaction when each new (child) node determines how it must interact based on observing the previously existing network (considered to be the environment, with behavior independent of the new node).

Interactive training takes advantage of potential coordination that can exist on a constant basis among all classifiers (networks), not just on an iterative basis where a single node or network is trained at a time, then incorporated into the existing system. Possibly, interactive behavior could create reactive oscillations between networks as they each update their behavior based on how the other net is currently behaving, back and forth (similar to the herd effect).

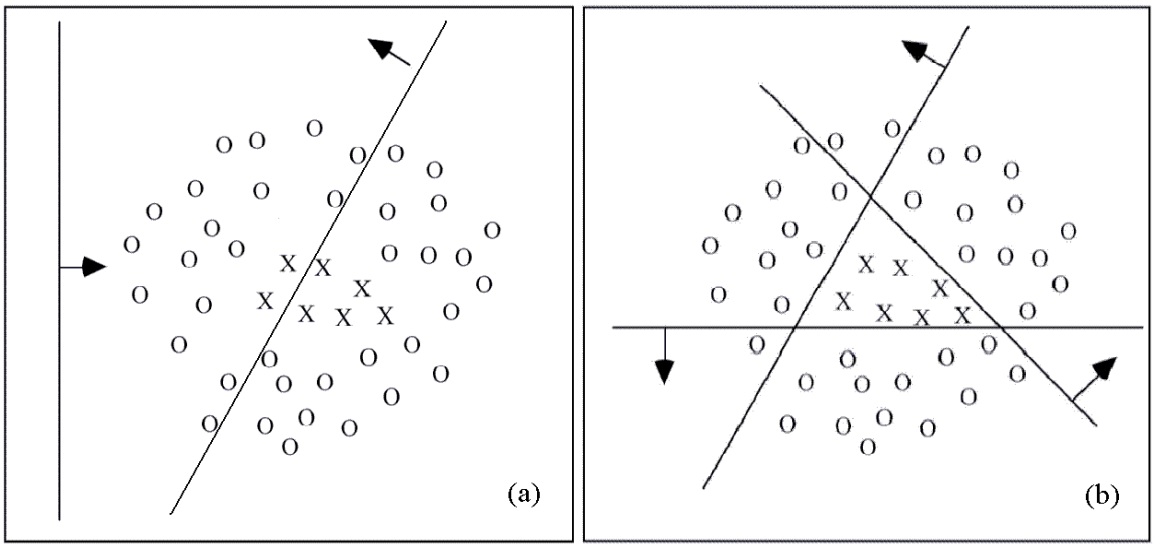

Figure 11. Decision surfaces for a simple training set.

The importance of interaction can be seen in the following example. Consider the simple training set with two possible hyperplanes shown in Figure 11a. The hyperplanes can be considered decision surfaces, with the arrow indicating which side of the hyperplane for which examples are classified as O's. Since individual perceptrons only provide solutions to linearly separable data, if a simple perceptron is trained on such data, the resulting decision surface will look like the leftmost hyperplane. This hyperplane maximizes classification accuracy. Observe that such a solution, although correctly classifying all of the O's, will incorrectly classify all of the X's.

When combining several such perceptrons into an ensemble, they are combined using an arbitrary linear combination. In this case, the ensemble is incapable of improving the accuracy of the individual perceptrons. Since all perceptrons are generalizing equivalently at baseline accuracy, the combined decision surface effectively becomes an

average of all the decision surfaces scattered around the perimeter of the training set. This puts the ultimate decision surface somewhere in the middle, resulting in generalization accuracy of about 50%, lower than what each single perceptron achieves.

Now when perceptrons are allowed to learn interactively in a fashion that they may coordinate their results during training, much better results are obtainable (see Figure 11b). Coordinated perceptrons do not have the problem illustrated in Figure 11a where they remain outside of the data cloud. If their outputs are put through a squashing function and summed, they can perform as a universal function approximator [Hor89]. They can fit any data distribution, provided there are enough of them. Observe that this interaction describes the behavior of the MLP [Rum85]. The MLP gains its expressive power by allowing perceptrons to interact and learn together as members of a hidden layer.

7.1.4 Expressiveness of multi-layer perceptrons

The multi-layer perceptron (MLP) [Rum85] consists of two or more layers of perceptrons (see Figure 12). Analogously to the perceptron, the weights of an MLP are updated through gradient descent under the "generalized delta rule." The error signal is propagated back from the final (output) network layer back to the initial (input) layer, a process known as error *backpropagation*. MLPs are proven to be universal function approximators [Hor89].

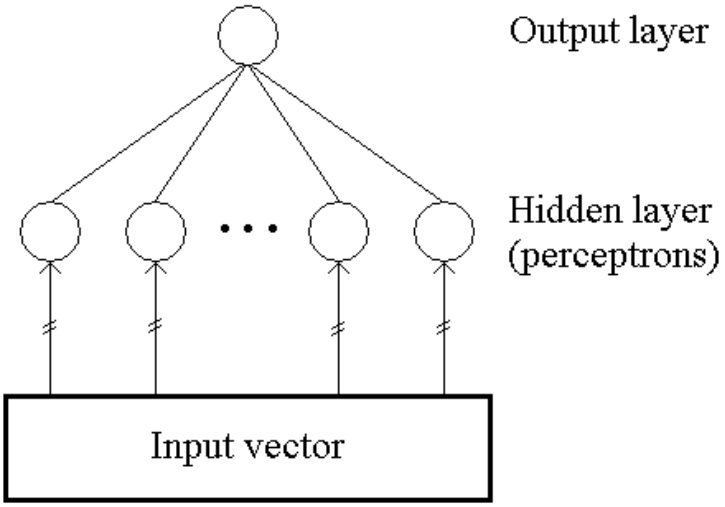

Figure 12. Feed-forward multi-layer perceptron.

7.1.5 Interactive training architecture

Interactive training with multiple networks provides a logical extension of the standard multi-layer perceptron (MLP). In an MLP, an input vector fans out to a layer of hidden nodes. The output of the hidden nodes is consolidated into one or more output nodes, generally with sigmoid activation functions. Under IT, a similar architecture is utilized (see Figure 13). However, there are two crucial differences in the implementation.

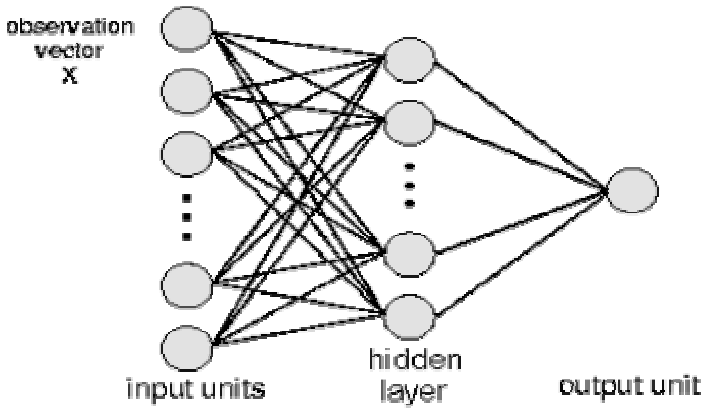

Figure 13. Interactive training network.

First, rather than being comprised of perceptrons, the hidden layer of an IT model is comprised of MLPs (individual networks). Each MLP serves to specialize in learning a certain area of the problem space. For instance, in learning a multi-class data distribution, each MLP can be responsible for learning one of the classes.

Second, the output node is essentially another classifier stacked onto the MLP vector. For a given sample, the feature vector is presented to all MLPs in the network's hidden layer. Each MLP sends an output to the classifier. A simple classifier is the winner-take-all function. The sample is classified correctly if an MLP corresponding to the target output class propagates the highest activation to the WTA node.

7.2 Improved objective function

The philosophy of lazy training has ties to multi-agent learning. Stirling points out that typically, optimal single agent solutions are not jointly optimal in multi-agent settings [Sti99]. This is demonstrated for neural network architecture optimization schemes for voting [And01a]. It is shown that the optimal network architecture selected for a single network model is typically not the optimal architecture when many such networks are combined with voting techniques such as bagging. The optimal architecture when a single network is used is often much less complex than the optimal network architecture of several networks being combined into a voting ensemble.

Coordination among agents occurs if members of a multi-agent system use information concerning the existence, decisions, or perceived decision-making strategies of other

agents. Coordination should provide a means of decision-making for each agent in the system as a function of both the agent's individual agenda and its relationships to the other members of the system [Sti96]. Coordinating means enabling a network to update its parameters based not only on its behavior, but the behavior of the other networks in the system. A network's complexity must be necessarily restrained when using standard learning heuristics to avoid overfitting. Coordinating network behavior using lazy training allows the use of a more complex model without engendering overfit.

Lazy training provides coordination among multiple single-output networks, or among output nodes in a multi-output network. Lazy training illustrates the principle of *satisficing* (see [Sim59]), where an aspiration level is specified, such that once that level is met, the corresponding solution is deemed adequate. Lazy 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 each network's exactness and the classification accuracy of the entire 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. Recall that classification generalization is the ultimate goal of learning, and it does not ultimately matter how close an output fires to zero or one. Thus, by relaxing the constraint of optimal credibility, resultant rejectability is reduced.

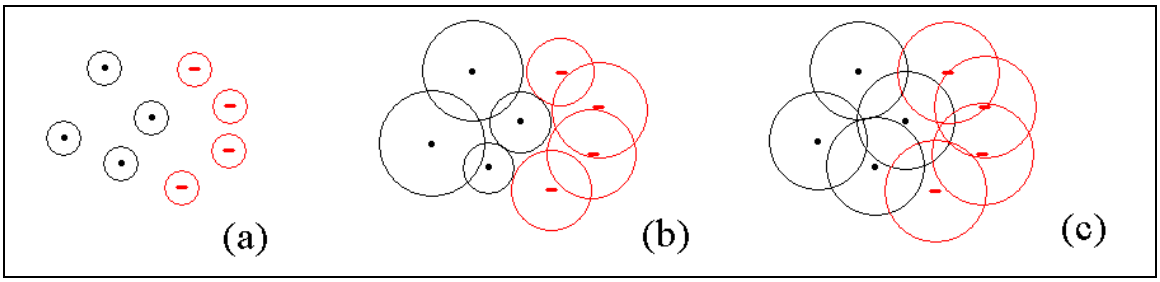

Figure 14. RBF networks with differing node variances.

Consider an RBF network where the node variance *r* of gaussian spheres of influence is a parameter. Choosing an *r* that is too small (Figure 14a) or too large (Figure 14c) often leads to poor generalization. An *r* just right leads to good generalization (see [Barn91] for an elaboration). An *r* too small is akin to overfitting, and too large to underfitting. A proper value for *r* provides a good overall solution, but does not guarantee that certain parts of the space are overfit, and others underfit. The intuition behind lazy training is that it attempts to find the balance between the two extremes on a local level. It effectively adds as much constraint as needed (by determining *r*) for a correct solution, but no more. It also allows *r* to be variable for each node (Figure 14b). Furthermore, as the maximum class output is the deciding factor, in practice variances can overlap, as long as the target class' influence is stronger for all patterns.

Chapter 8

Conclusion and Future Work

Lazy training 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 real world data sets. On UCI MLDR problems there was an average increase in accuracy from 90.7% for optimized SSE networks to 92.1% for lazy training performing 10-fold stratified cross-validation. Similarly, there was a drop in error from 2.14% to 0.89% on a large OCR data set.

There are many directions that future research on lazy training can take. The effect of modifying the upper and lower target bounds and error margin will be considered. It is natural to consider implementing dynamic versions of these parameters that change over time or using values local to each instance in the training set. Further studies of the effect of network size on lazy training behavior and classification accuracy will be done. The effect of lazy training in more sophisticated classification systems, where classification functions more complex than WTA are used, such as stacking a perceptron, MLP, or rule-based or search system onto the IT network [Rim00c], will be investigated.

We will explore the advantages of lazy training over independent learning on different network models, such as single-output networks on two-class problems, and multi-output networks (one output per class). Various styles of network interaction will be considered, including: (1) between dual (complementary) networks for each class (two per class), (2) among redundant networks (many per class), (3) within a lazy-trained multi-output network (one output per class).

Lazy training will be combined with many other learning enhancements, such as weight decay to produce even "simpler" solutions, RPROP [Rie93] or Quickprop [Fah88] to improve the rate of convergence, *speed training* [Rim00b] to speed up learning, methods for growing and pruning networks, forming network hierarchies, and so forth. Lazy training variants will also be considered for batch learning.

It has been observed that classification errors between standard and lazy trained networks are highly uncorrelated. Ensembles combining standard and lazy trained networks will be analyzed with the expectation that this will further reduce test error.

Bibliography

[Aha97] David W. Aha, editor, *Lazy Learning*, Kluwer Academic Publishers, Dordrecht, May 1997.

[And95] Andersen, Tim and Tony R. Martinez, "A Provably Convergent Dynamic Training Method for Multilayer Perceptron Networks," Proceedings of the *2nd International Symposium on Neuroinformatics and Neurocomputers*, pp. 77-84, 1995.

[And96] Andersen, Tim and Tony R. Martinez, "Using Multiple Node Types to Improve the Performance of DMP (Dynamic Multilayer Perceptron)," Proceedings of the *IASTED International Conference on Artificial Intelligence, Expert Systems and Neural Networks*, pp. 249-252, 1996.

[And99a] 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.

[And99b] 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.

[And01a] Andersen, Tim, Rimer, Michael and Martinez, Tony, "Optimal Artificial Neural Network Architecture Selection for Voting," Proceedings of the *IEEE International Joint Conference on Neural Networks IJCNN'01*, pp. 790-795, 2001.

[And01b] Andersen, Tim and Martinez, Tony, "DMP3: A Dynamic Multilayer Perceptron Construction Algorithm," *International Journal of Neural Systems*, vol. 11, No. 2 (2001), pp. 145-165.

[Barn91] Barnard, Etienne. "Performance and Generalization of the Classification Figure of Merit Criterion Function," *IEEE Transactions on Neural Networks*, vol. 2, no. 2, March 1991, pp. 322-325.

[Bart98] 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*, vol. 44 (2), 525--536, 1998.

[Bau99] Eric Bauer and Ron Kohavi, "An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants," *Machine Learning*, vol. 36, no. 1-2, pp. 105-139, 1999.

[Bia98] M. Bianchini, P. Frasconi, M. Gori, and M. Maggini, "Optimal learning in artificial neural networks: A theoretical view," in *Neural Network Systems Techniques and Applications* (C. Leondes, ed.), ch. 1, pp. 1-51, Academic-Press, 1998.

[Bla98] Blake, C.L. & Merz, C.J. (1998). UCI Machine Learning Repository, http://www.ics.uci.edu/~mlearn/MLRepository.html. Irvine, CA: University of California, Department of Information and Computer Science.

[Car93] Caruana, Rich, "Multitask Connectionist Learning," Proceedings of the *1993 Connectionist Models Summer School*, pp. 372-379, 1993.

[Car94] Caruana, Rich, "Learning Many Related Tasks at the Same Time With Backpropagation," *Advances in Neural Information Processing Systems 7* (Proceedings of NIPS*94), pp. 657-664, 1995.

[Car95] Rich Caruana, Shummet Baluja, and Tom Mitchell, "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), 1996.

[Car97] Caruana, Rich, "*Multitask Learning*," Ph.D. Thesis, School of Computer Science, CMU, 1997.

[Cas93] Castellano, G., A. M. Fanelli and M. Pelillo, "An empirical comparison of node pruning methods for layered feed-forward neural networks," Proceedings of the *IJCNN'93-1993 International Joint Conference on Neural Networks*, Nagoya, Japan, 1993, pp. 321-326.

[Cas97] Castellano, G., A.M. Fanelli, M. Pelillo, "An iterative pruning algorithm for feed-forward neural networks," *IEEE Transactions on Neural Networks*, vol. 8 (3), May 1997, pp. 519-531.

[Cha97a] Chakraborty, Goutam, Masayuki Sawada and Shoichi Noguchi, "Combining Local Representative Networks to Improve Learning in Complex Nonlinear Learning Systems," *IEICE Trans. Fundamentals*, vol. E80-A, No. 9, Sep. 1997, pp. 1630-33.

[Cha97b] Chakraborty B., Y. Sawada and G. Chakraborty, "Layered fractal neural net: computational performance as a classifier," *Knowledge-based Systems*, vol. 10, 1997, pp. 177-182.

[Dud99] Richard O. Duda, Peter E. Hart, David G. Stork, *Pattern Classification*, 2nd edition, 1999. Wiley, John & Sons, Inc.

[Fah88] Fahlman, S.E., "Faster-learning Variations on Back-propagation: An Empirical Study," Proceedings of the *1988 Connectionist Models Summer School*, Morgan Kaufmann.

[Fah90] Fahlman, Scott E. and Lebiere, Christian, "The Cascade-Correlation Learning Architecture," in *Advances in Neural Information Processing Systems 2*, David S. Touretsky, ed., Morgan Kaufmann, San Mateo, California, 1990, pp. 524-532.

[Fri97] Friedman, J.H. "On Bias, Variance, 0/1-Loss, and the Curse-of-Dimensionality," *Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 55-77. Kluwer Academic Publishers. (1997)

[Gem92] S. Geman and E. Bienenstock, "Neural Networks and the Bias/Variance Dilemma," *Neural Computation,* vol. 4, pp. 1-58, 1992.

[Goo00] Goodrich, Michael A., Wynn C. Stirling, and Richard L. Frost, "Multi-agent Satisficing Choice: Concepts and Case Study," 2000, Submitted to *Autonomous Agents and Multi-agent Systems Journal*.

[Gor88] Gorman, R. P., and Sejnowski, T. J., "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets," *Neural Networks*, vol. 1, pp. 75-89, 1988.

[Ham89] Hampshire, John and Waibel, A., "The meta-pi network: Building distributed knowledge representations for robust pattern recognition," Tech Rep. CMU-CS-89-166, Carnegie Mellon University, Pittsburgh, PA. 1989.

[Ham90] 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.

[Hor89] Hornik, Kurt, Maxwell Stinchcombe and Halbert White. "Multi-layer neural networks are universal function approximators," *Neural Networks*, vol. 2, 1989, pp. 359-66.

[Jac88] Jacobs, Robert A., "Increased Rates of Convergence Through Learning Rate Adaptation," *Neural Networks*, vol. 1, pp. 295-307.

[Jac91] Jacobs, Robert A., Michael Jordan, Steven J. Nowlan, Geoffrey E. Hinton, "Adaptive Mixtures of Local Experts," *Neural Computation*, vol. 3, pp. 79-87 (1991).

[Joo98] M. Joost and W. Schiffmann, "Speeding up Backpropagation Algorithms by using Cross--Entropy combined with Pattern Normalization," *International Journal of Uncertainity, Fuzziness and Knowledge--based Systems* (IJUFKS), vol. 6, no. 2, pp. 117-126, 1998.

[Law00] Steve Lawrence and C. Lee Giles, "Overfitting and Neural Networks: Conjugate Gradient and Backpropagation," *International Joint Conference on Neural Networks*, Como, Italy, July 24–27, pp. 114–119, 2000.

[Leb93] M. LeBlanc and R. Tibshirani, "Combining estimates in regression and classification," *NeuroProse*, 1993.

[Lec90] LeCun, Y., Denker, J. and Solla, S., "Optimal Brain Damage," *Advances in Neural Information Processing Systems (NIPS) 2*, D. S. Touretzky, editor, San Mateo, Morgan Kaufmann Publishers Inc., pp. 598-605, 1990.

[Liu99a] Liu, Yong and Yao, Xin, "Simultaneous Training of Negatively Correlated Neural Networks in an Ensemble," *IEEE Transactions on Systems, Man and Cybernetics*, Part B: Cybernetics, vol. 29, no. 6, pp. 716-725, December 1999.

[Liu99b] Liu, Yong and Yao, Xin, "Ensemble learning via negative correlation," *Neural Networks*, vol. 12, no. 10, pp. 1399-1404, December 1999.

[Mac97] Maclin, R and Opitz, D., "An empirical evaluation of bagging and boosting," *The Fourteenth National Conference on Artificial Intelligence*, 1997.

[Min69] Minsky, Marvin L. and Papert, Seymour S., *Perceptrons*: *An Introduction to Computational Geometry*, MIT Press, Cambridge, MA 1969.

[Mit97] Mitchell, Tom. *Machine Learning.* McGraw-Hill Companies, Inc., Boston, 1997.

[Qia99] Qian, Ning, "On the Momentum Term in Gradient Descent Algorithms," *Neural Networks*, vol. 12, no. 1, pp. 145-151, 1999.

[Rie93] 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.

[Rim00a] 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.

[Rim00b] Rimer, Michael E., Anderson, Timothy L. and Martinez, Tony R., "Speed Training: Improving Learning Speed for Large Data Sets," Proceedings of the *IEEE International Joint Conference on Neural Networks IJCNN'01*, pp. 2662-2666, 2001.

[Rim00c] Rimer, Michael E., Martinez, Tony R., and Wilson, D. Randall, "Improving Speech Recognition Learning through Lazy Training," to appear in to appear in the *IEEE International Joint Conference on Neural Networks IJCNN'02*, 2002.

[Ros58] F. Rosenblatt. “The perceptron: A probabilistic model for information storage and organization in the brain,” *Psychological Review*, 65:386-408, 1958.

[Rum85] 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.

[Sch93] Schiffmann, W., Joost, M. and Werner, R. “Comparison of Optimized Backpropagation Algorithms,” *Artificial Neural Networks*, European Symposium, Brussels, 1993.

[Sha96] Sharkey, A. “On Combining Artificial Neural Nets,” *Connection Science*, 1996, vol. 8(3-4), pp. 299-313.

[Sim59] Simon, Herbert. “Theories of decision-making in economics and behavioral science,” *American Economic Review*, XLIX (1959), 253.

[Sti96] W. C. Stirling, M. A. Goodrich, R. L. Frost. “Toward a theoretical foundation for multi-agent coordinated decisions,” in *Proceedings of the Second International Conference on Multi-Agent Systems*, pages 345-352, Kyoto, Japan, 1996.

[Sti99] W. C. Stirling and M. A. Goodrich. “Satisficing games,” *Information Sciences*, 114:255-280, March 1999.

[Sto93] D. Stork and B. Hassibi, “Second order derivatives for network pruning: Optimal Brain Surgeon.” In T. J. Sejnowski, G. E. Hinton and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems (NIPS) 5*, pp. 164-171, San Mateo, 1993. Morgan Kaufmann Publishers Inc.

[Tol90] Tollenaere, Tom, “SuperSAB: Fast Adaptive Back Propagation with Good Scaling Properties,” *Neural Networks*, vol. 3, pp. 561-573, 1990.

[Wan94] 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, pp. 303-310. Morgan Kaufmann, San Francisco.

[Wid60] Widrow, B. and Hoff, M. E, “Adaptive switching circuits,” In *IRE WESTCON Connection Record*, vol. 4 (1960), pp. 96-104.

[Weg97] Wegner, Peter. “Why Interaction is more powerful than algorithms,” *Communications of the ACM*, May 1997.

[Weg99] Wegner, Peter and Goldin, Dina, “Interaction, Computability and Church’s Thesis,” 1999. Accepted to the *British Computer Journal*.

[Wei99] Gerhard Weiss, editor. *Multi-agent Systems, A Modern Approach to Distributed Artificial Intelligence*. 1999. MIT Press, Cambridge, Massachusetts.

[Wer88] Werbos, P., "Backpropagation: Past and future," *Proceedings of the IEEE International Conference on Neural Networks*, pp. 343-353. IEEE Press, 1988.

[Wol92] Wolpert, D.H. (1992), "Stacked Generalization," *Neural Networks*, vol. 5, pp. 241-259, Pergamon Press.

[Zar95] Zarndt, Frederick, "A Comprehensive Case Study: An Examination of Machine Learning and Connectionist Algorithms," Masters Thesis, Brigham Young University, 1995.