2007-03-12

Improving Machine Learning Through Oracle Learning

Joshua Ephraim Menke
Brigham Young University - Provo

IMPROVING MACHINE LEARNING THROUGH ORACLE LEARNING

by

Joshua E. Menke

A dissertation submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

Brigham Young University

April 2007

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a dissertation submitted by

Joshua E. Menke

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

Date	Tony R. Martinez, Chair
Date	Dan Ventura
Date	Kevin Seppi
Date	Thomas W. Sederberg
Date	Mark Clement

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the dissertation of Joshua E. Menke 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	Parris K. Egbert Graduate Coordinator
Accepted for the College	Thomas W. Sederberg Associate Dean, College of Physical and Mathematical Sciences

ABSTRACT

IMPROVING MACHINE LEARNING THROUGH ORACLE LEARNING

Joshua E. Menke

Department of Computer Science

Doctor of Philosophy

The following dissertation presents a new paradigm for improving the training of machine learning algorithms, *oracle learning*. The main idea in *oracle learning* is that instead of training directly on a set of data, a learning model is trained to approximate a given *oracle's* behavior on a set of data. This can be beneficial in situations where it is easier to obtain an oracle than it is to use it at application time. It is shown that oracle learning can be applied to more effectively reduce the size of artificial neural networks, to more efficiently take advantage of domain experts by approximating them, and to adapt a problem more effectively to a machine learning algorithm.

ACKNOWLEDGMENTS

I would first like to thank my wonderful wife Maren, who stuck by me and would never consider me leaving without finishing this dissertation. She deserves special credit for listening to ideas in a field that was of little interest to her.

I would also like to thank my advisor, Dr. Tony Martinez, whose encouragement and direction helped me transition from being a college graduate to enjoying research. I am also grateful to the rest of my committee for allowing me to consult with them when I was working on ideas that more closely matched their expertise.

I thank my Heavenly Father who helped me repeatedly by giving me insight as to how to apply and develop the ideas in this dissertation when my own abilities came up short.

I also thank Dr. Shane Reese, who gave me last minute advisement as a statistics professor even though he was not on my committee, and whose guidance is largely responsible for the results of part 3 of this dissertation.

I am also grateful to my fellow graduate students, especially those I worked closely with in my lab. They were always there to bounce ideas off of, and to listen while I worked out the answers on my own.

I also need to thank the members of the Beginner's Park 3 gaming community, who patiently helped me test many of the ideas in this dissertation, and provided needed criticism when my experiments negatively impacted their gameplay.

Contents

Part I

Introduction

Part I provides a brief overview of the parts of the dissertation.

Chapter 1 gives an overview of the dissertation by introducing oracle learning. It summarizes how oracle learning can be used for reducing the size of artificial neural networks, for approximating domain experts, and how it can be used to adapt a given problem to a machine learning algorithm instead of only adapting the learner to the problem. More detail on oracle learning is presented in part II. Since the body of the dissertation is composed of publications accepted or submitted to refereed journals or conferences, the end of chapter 1 lists the publications that correspond to chapters 2–9.

In addition to the work done in oracle learning, some additional research was conducted in the area of paired comparisons. These papers are introduced and given at the end of the dissertation in part III.

Chapter 1

Dissertation Overview

The main part of this dissertation (Part II) introduces oracle learning and how it can be applied to improve machine learning. Machine learning algorithms are designed to infer relationships from data sets. This process is often called "training." For example, a common application of machine learning algorithms is to use them to train classifiers. A trained classifier should be able to take the features of a given data point as input and return the "class" of that data point. If a given classifier were trained on data that gave examples of apples and oranges, it should be able to determine, given a new data point, whether it is an apple or an orange.

The main idea in oracle learning is that instead of training directly on a set of data, a learning model is trained to approximate a given *oracle's* behavior on a set of data. The oracle can be another learning model that has already been trained on the data, or it can be any given functional mapping $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ where n is the number of inputs to both the mapping and the *oracle-trained model* (OTM), and m is the number of outputs from both. The main difference with oracle learning is that the OTM trains on a training set whose targets have been relabeled by the oracle instead of training with the original training set labeling. Having an oracle to label data means that previously unlabeled data can also be used to augment the relabeled training set. The key to oracle learning's success is that it attempts to use a training set that fits the observed distribution of the given problem to accurately approximate

the oracle on those sections of the input space that are most relevant in real-world situations.

One use of oracle learning allows large ANNs that are computationally intensive in terms of both space and time to be reduced in size without losing as significant amount of accuracy. This allows ANNs to be used more effectively in the ever growing sector of embedded devices which include, for example, PDAs and cell phones. In chapter 2, small ANNs are trained to approximate larger ANNs instead of being trained directly on the data. In addition, the smaller ANNs are trained on previously unlabeled data since the larger ANNs can serve as *oracle ANNs* to label data that did not originally have labels. Using oracle learning to reduce the size of these ANNs resulted in a 15% decrease in error over standard training and maintained a significant portion of the oracles' accuracy while being as small as 6% of the oracles' size.

In chapter 3, oracle learning is used to approximate multiple domain experts with a single ANN. For a given application, higher generalization accuracy can be obtained by training separate learning models as "experts" over specific domains. For example, given an application where it is common to observe at least two varying levels of noise, clean and noisy, one solution is to train a single classifier on both clean and noisy data. It is possible to achieve better accuracy by training one classifier on only noisy data and one classifier on only clean data, and then choosing between them during classification depending on the environment. The clean and noisy domain experts will have higher accuracy on their respective domains than a classifier trained on a mix of both clean and noisy data. Unfortunately, it is difficult to know beforehand whether a given data point belongs to the clean or noisy section of the data, and therefore it is difficult to know whether to use the clean or noisy domain expert. Chapter 3 presents the *bestnets* method which uses oracle learning to approximate the behavior of both the clean and noisy domain experts with a single learning model. On a set of both noisy and clean optical character recognition data, using oracle learning

to approximate the domain experts resulted in a statistically significant improvement ($p < 0.0001$) over standard training on the mixed data.

It is well known that no machine learning algorithm does well over all functions [74], however chapter 4 uses oracle learning to show that it may be possible to adapt a given function to better fit a given learning algorithm. Instead of only training the learner on the problem, chapter 4 shows that the problem can be "trained" on the learner simultaneously, in order to improve performance. Adapting the problem to the learner may result in an equivalent function that is "easier" for a given algorithm to learn. A general approach for adapting problems to their learners is proposed in this chapter. This method takes an arbitrary data set and uses oracle learning to modify that data set to better fit the learning algorithm. The results are that the learning algorithm attains higher classification accuracy on a test set taken from the original data set. This method for target relabeling combines *self-oracle learning* (SOL) and ANN confidence measures. SOL is a proof of concept method to demonstrate the potential for adapting problems to the learner. The work in chapter 4 combines SOL with *Confidence-based Target Relabeling* (CTR). CTR is a method for estimating the confidence that an ANN has in its given outputs. SOL can use the results of CTR to modify the oracle labels based on how confident the ANN is in its outputs. Applying *Self-Oracle Learning with Confidence-based Target Relabeling* (SOL-CTR) over 41 data sets consistently results in a statistically significant ($p < 0.05$) improvement in accuracy over 0/1 targets on data sets containing over 10,000 training examples.

The final chapter in part II serves to answer additional questions about the usefulness of oracle learning in general. It compares the oracle learning methods in chapters 2 and 4 to standard training, weight decay, and basic self-oracle learning (SOL) on situations where a smaller ANN is and is not desired, and in situations where unlabeled data is and is not available. Performance is measured across 35 datasets including a larger automated speech recognition (ASR), two optical character

recognition (OCR) data sets, and 32 sets from the UCI Machine Learning Database (MLDB) repository. The results show that in most cases SOL-CTR is the preferred method and that it either yields a statistically significant improvement over each other method, or it gives results that are never worse than the others. Exceptions to this occur when shrinking the size of an ANN for ASR and the UCI sets in the presence of unlabeled data in which case using the oracle learning method in chapter 2 is preferable.

The chapters in part II go into further detail on the specifics of oracle learning and how it is applied.

Part III of the dissertation adds a few additional papers that are not directly related to oracle learning, but instead to the problem of paired-comparisons. These sections are introduced further in part III.

1.1 Publications

Chapters 2–4 and chapters 6–9 are based on a collection of papers that have either been published or submitted for publications in refereed journals or conferences. Following is a list of references for these publications in the order they appear in this dissertation.

II. Oracle Learning

Joshua E. Menke and Tony R. Martinez. Artificial neural network reduction through oracle learning. Submitted to *Neural Processing Letters*.

Joshua Menke and Tony R. Martinez. Domain expert approximation through oracle learning. In *Proceedings of the 13th European Symposium on Artificial Neural Networks (ESANN 2005)*. 2005.

Joshua E. Menke and Tony R. Martinez. Improving machine learning by adapting the problem to the learner. Submitted to the *International Journal of Neural Systems* 2006.

III. Paired Comparisons

Joshua Menke and Tony R. Martinez. Using permutations instead of student's t distribution for p-values in paired-difference algorithm comparisons. In *Proceedings of the 2004 IEEE Joint Conference on Neural Networks IJCNN'04*. 2004.

Joshua E. Menke and Tony R. Martinez. A Bradley-Terry artificial neural network model for individual ratings in group competitions. To appear in *Neural Computing and Applications*, 2007.

Joshua E. Menke, C. Shane Reese, and Tony R. Martinez. Hierarchical models for estimating individual ratings from group competitions. In prepartion for the *Journal of the American Statistical Association*.

Joshua E. Menke, C. Shane Resse, and Tony R. Martinez. A method for estimating individual ratings from group competitions in real-time. In preparation for the *Journal of Applied Statistics*, 2006.

Part II

Oracle Learning

The following chapters present oracle learning and how it can be applied to improve machine learning.

Chapter 2 shows how oracle learning can successfully reduce the size of artificial neural networks.

Chapter 3 gives an additional application of oracle learning. Here it is applied to approximate the performance of multiple domain experts.

In chapter 4, it is shown that oracle learning can be applied to improve machine learning in general by better adapting a given problem to a given learner.

Chapter 5 serves to answer additional questions about when to apply the oracle learning methods given in chapters 2 and 4.

Chapter 2

Artificial Neural Network Reduction Through Oracle Learning

Abstract

Often the best model to solve a real-world problem is relatively complex. This paper presents *oracle learning*, a method using a larger model as an oracle to train a smaller model on unlabeled data in order to obtain (1) a smaller acceptable model and (2) improved results over standard training methods on a similarly sized smaller model. In particular, this paper looks at oracle learning as applied to multi-layer perceptrons trained using standard backpropagation. Using multi-layer perceptrons for both the larger and smaller models, oracle learning obtains a 15.16% average decrease in error over direct training while retaining 99.64% of the initial oracle accuracy on automatic spoken digit recognition with networks on average only 7% of the original size. For optical character recognition, oracle learning results in neural networks 6% of the original size that yield a 11.40% average decrease in error over direct training while maintaining 98.95% of the initial oracle accuracy. Analysis of the results suggest oracle learning is especially appropriate when either the size of the final model is relatively small or when the amount of available labeled data is small.

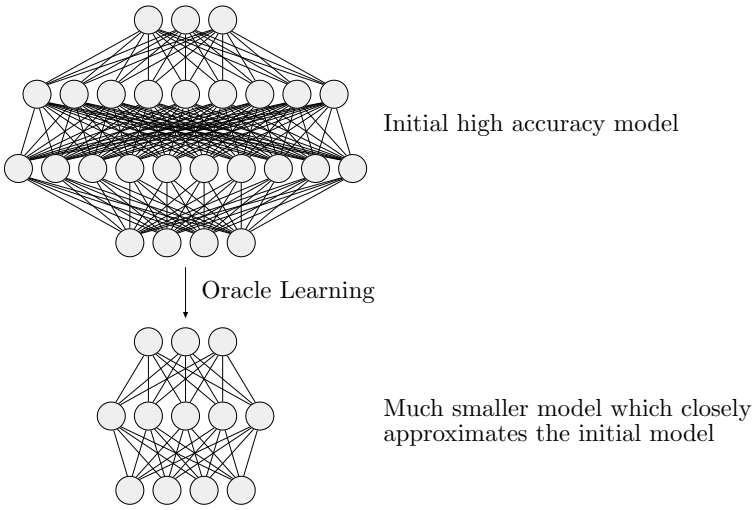

Figure 2.1: Using oracle learning to reduce the size of a multi-layer ANN.

2.1 Introduction

As Le Cun et. al observed [43], often the best *artificial neural network* (ANN) to solve a real-world problem is relatively complex. They point to the large ANNs used by Waibel for phoneme recognition [72] and the ANNs of LeCun et. al with handwritten character recognition [42]. "As applications become more complex, the networks will presumably become even larger and more structured" [43]. The following research presents the *oracle learning* algorithm, a training method that takes a large, highly accurate ANN and uses it to create a new ANN which is (1) much smaller, (2) still maintains an acceptable degree of accuracy, and (3) provides improved results over standard training methods (figure 2.1). Designing an ANN for a given application requires first determining the optimal size for the ANN in terms of accuracy on a test set, usually by increasing its size until there is no longer a significant decrease in error. Once found, this preferred size is often relatively large for more complex problems. One method to reduce ANN size is to just train a smaller ANN using standard methods. However, using ANNs smaller than the optimal size results in a

decrease in accuracy. The goal of oracle learning is to create smaller ANNs that are more accurate than can be directly obtained using standard training methods.

As an example consider designing an ANN for optical character recognition in a small, hand-held scanner. The ANN has to be small, fast, and accurate. Now suppose the most accurate digit recognizing ANN given the available training data has 2048 hidden nodes, but the resources on the scanner allow for only 64 hidden nodes. One solution is to train a 64 hidden node ANN using standard methods, resulting in a compromise of significantly reduced accuracy for a smaller size. This research demonstrates that applying oracle learning to the same problem results in a 64 hidden node ANN that does not suffer from nearly as significant a decrease in accuracy. Oracle learning uses the original 2048 hidden node ANN as an oracle to create as much training data as necessary using unlabeled character data. The oracle labeled data is then used to train a 64 hidden node ANN to exactly mimic the outputs of the 2048 hidden node ANN. The results in section 2.5 show the oracle learning ANN retains 98.9% of the 2048 hidden node ANN's accuracy on average, while being 3.13% of the size. The resulting *oracle-trained network* (OTN) is 17.67% more accurate on average than the standard trained 64 hidden node ANN. Analysis of the results suggest oracle learning is especially appropriate when either the size of the final model is relatively small or when the amount of available labeled data is small.

Although the previous example deals exclusively with ANNs, both the oracle model and the *oracle-trained model* (OTM) can be any machine learning model (e.g. an ANN, a nearest neighbor model, a bayesian learner, etc.) as long as the oracle model is a functional mapping $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ where n is the number of inputs to both the mapping and the OTM, and m is the number of outputs from both. Note that if the outputs of the oracle are class labels instead of continuous values per class, they can still be represented in $\mathbb{R}^m$ by encoding them as discrete targets, or any other

encoding appropriate for the OTM. The same unlabeled data is fed into both the oracle and the OTM, and the error used to train the OTM is the oracle's output minus the OTM's output. Thus the OTM learns to minimize its differences with the oracle on the unlabeled data set. The unlabeled data set must be drawn from the same distribution that the smaller model will be used to classify. How this is done is discussed in section 2.3.2. Since the following research uses multilayer feed-forward ANNs with a single-hidden layer as both oracles and OTMs, the rest of the paper describes oracle learning in terms of ANNs. An ANN used as an oracle is referred to as an *oracle ANN* (a standard backpropagation trained ANN used as an oracle). Note that since the goal of the oracle learning is to match and not outperform the oracle, the theoretical constraints are no different than those of standard backpropagation training.

Oracle learning can be used for more than just model size reduction. An interesting area where we have already obtained promising results is approximating a set of ANNs that are "experts" over specific parts of a given function's domain. A common solution to learning where there are natural divisions in a function's domain is to train a single ANN over the entire function. This single ANN is often less accurate on a given sub-domain of the function than an ANN trained only on that sub-domain. The ideal would be to train a separate ANN on each natural division of the function's domain and use the appropriate ANN for a given pattern. Unfortunately, it is not always trivial to determine where a given pattern lies in a function's domain. We propose the *bestnets* method which uses oracle learning to reduce the multiple-ANN solution to a single OTN. Each of the original ANNs is used as an oracle to label those parts of the training set that correspond to that ANN's "expertise." The resulting OTN attempts to achieve similar performance to the original set of ANNs without needing any preprocessing to determine where a given pattern lies in the function's domain. One application to which we have successfully applied the bestnets method is

to improve accuracy when training on data with varying levels of noise. Section 2.5.4 gives initial results on an experiment with clean and noisy *optical character recognition* (OCR) data. The results show that the bestnets-trained ANN is statistically more accurate than an ANN trained directly on the entire function. The resulting p-value is less than 0.0001—meaning the test is more than 99.99% confident that the bestnets-trained ANN is more accurate than the ANN trained on the entire function.

2.2 Background

Three research areas related to oracle learning are:

1. Model Approximation
2. Decreasing ANN Size
3. Use of Unlabeled Data

The idea of approximating a model is not new. Domingos [20] used Quinlan's C4.5 decision tree approach [61] to approximate a bagging ensemble [8] and Zeng and Martinez [78] used an ANN to approximate a similar ensemble. Craven and Shavlik used a similar approximating method to extract rules [14] and trees [15] from ANNs. Domingos and Craven and Shavlik used their ensembles to generate training data where the targets were represented as either being the correct class or not. Zeng and Martinez used a target vector containing the exact probabilities output by the ensemble for each class. The following research also uses vectored targets similar to Zeng and Martinez since Zeng's results support the hypothesis that vectored targets "capture richer information about the decision making process ..." [78]. While previous research has focused on either extracting information from ANNs [14, 15] or using statistically generated data for training [20, 78], the novel approach presented here is that available, unlabeled data be labeled using the more accurate model as an oracle.

One goal of oracle learning is to produce a smaller ANN. Pruning [63] is another method used to reduce ANN size. Section 2.5.3 presents detailed results of comparing oracle learning to pruning for reducing ANN size. The results show that pruning is less effective than oracle learning in terms of both size and accuracy. In particular, oracle learning is more effective when reducing an initial model to a specified size.

Much work has recently been done using unlabeled data to improve the accuracy of classifiers [4, 56, 55, 29, 3]. The basic approach, often known as semi-supervised learning or boot-strapping, is to train on the labeled data, classify the unlabeled data, and then train using both the original and the relabeled data. This process is repeated multiple times until generalization accuracy stops increasing. The key differences between semi-supervised and oracle learning are enumerated in section 2.3.4. The main difference stems from the fact that semi-supervised learning is used to improve accuracy and outperform training with only labeled data, whereas oracle learning is used here only for model size reduction.

2.3 Oracle Learning: 3 Steps

2.3.1 Obtaining the Oracle

The primary component in oracle learning is the oracle itself. Since the accuracy of the oracle ANN directly influences the performance of the final, smaller ANN, the oracle must be the most accurate classifier available, regardless of complexity (number of hidden nodes). In the case of ANNs, the most accurate classifier is usually the largest ANN that improves over the next smallest ANN on a validation set. The only requirement is that the number and type of the inputs and the outputs of each ANN (the oracle and the OTN) match.

Notice that by this definition of how the oracle ANN is chosen, any smaller, standard-trained ANN must have a significantly lower accuracy. This means that if a

smaller OTN approximates the oracle such that their differences in accuracy become insignificant, the OTN will have a higher accuracy than any standard-trained ANN of its same size—regardless of the quality of the oracle.

The oracle should be chosen using a validation (hold-out) set (or similar method) in order to prevent over-fitting the data with the more complex model. As stated above, we chose the most accurate ANN improving over the next smallest ANN on a validation or hold-out set in order to prevent the larger ANN from over-fitting.

2.3.2 Labeling the Data

The key to the success of oracle learning is to obtain as much data as possible that ideally fits the distribution of the problem. There are several ways to approach this. Zeng and Martinez [78] use the statistical distribution of the training set to create data. However, the complexity of many applications makes accurate statistical data creation very difficult since the amount of data needed increases exponentially with the dimensionality of the input space. Another approach is to add random jitter to the training set according to some (a Gaussian) distribution. However, early experiments with the jitter approach did not yield promising results. The easiest way to fit the distribution is to have more real data. In many problems, like *automatic speech recognition* (ASR), labeled data is difficult to obtain whereas there are more than enough unlabeled real data that can be used for oracle learning. The oracle ANN can label as much of the data as necessary to train the OTN and therefore the OTN has access to an arbitrary amount of training data distributed as they are in the real world.

To label the data, this step creates a target vector $\mathbf{t^j} = \mathbf{t_1} \ldots \mathbf{t_n}$ for each input vector $\boldsymbol{x}^j$ where each t_i is equal to the oracle ANN's activation of output i given the j^{th} pattern in the data set, $\boldsymbol{x}^j$. Then, the final oracle learning data point contains

both $\boldsymbol{x}^j$ and $\boldsymbol{t}^j$. In order to create the labeled training points, each available pattern $\boldsymbol{x}^j$ is presented as a pattern to the oracle ANN which then returns the output vector $\boldsymbol{t}^j$. The final oracle learning training set then consists of the pairs $\boldsymbol{x}^1\boldsymbol{t}^1 \ldots \boldsymbol{x}^m\boldsymbol{t}^m$ for all m of the previously unlabeled data points.

Therefore, the data are labeled with the exact outputs of the oracle instead of just using the class information. Preliminary experimentation showed using the exact outputs yielded improved results over using only the class information. In addition, Zeng and Martinez [78] show improved results using exact outputs when approximating with ANNs. Current research is investigating the cause of this improvement. Also, since exact instead of $0-1$ labels are used, oracle learning is closer to function approximation than classification learning.

We submit that it is possible for the smaller OTN to fit its potentially more complex oracle because of the following:

1. Although the larger ANN can theoretically represent a more complex function, by using a validation set to prevent over-fit, the final function of the larger ANN may actually be simpler than is theoretically possible. Therefore, a smaller ANN may be able to represent the same function as the larger oracle ANN if trained properly.

2. Oracle learning may present a function that is equivalent in the classification sense, but easier for backpropagation to learn, and therefore a smaller ANN can still learn the function. It could be, for example, that using the exact oracle outputs instead of $0-1$ targets creates a function easier for backpropagation to learn. Caruana proves this is possible in his work on *RankProp* [9, 10, 11].

Notice that the goal of the OTN is *not* to outperform the oracle, but only to approximate it. As far as the OTN is concerned, the unlabeled data is normal labeled data and therefore the training process from the OTN point of view is no different than standard function approximation with backpropagation.

2.3.3 Training the OTN

For the final step, the OTN is trained using the data generated in step 2, utilizing the targets exactly as presented in the target vector. The OTN interprets each real-valued element of the target vector $\boldsymbol{t}^j$ as the correct output activation for the output node it represents given $\boldsymbol{x}^j$. The back-propagated error is therefore $t_i - o_i$ where t_i is the i^{th} element of the target vector $\boldsymbol{t}^j$ (and also the i^{th} output of the oracle ANN) and o_i is the output of node i. This error signal causes the outputs of the OTN to approach the target vectors of the oracle ANN on each data point as training continues.

As an example, the following vector represents the output vector $\boldsymbol{o}$ for the given input vector $\boldsymbol{x}$ of an oracle ANN and $\hat{\boldsymbol{o}}$ represents the output of the OTN. Notice the 4^{th} output is the highest and therefore the correct one as far as the oracle ANN is concerned.

$$\boldsymbol{o} = \langle 0.27, 0.34, 0.45, 0.89, 0.29 \rangle \tag{2.1}$$

Now suppose the OTN outputs the following vector:

$$\hat{\boldsymbol{o}} = \langle 0.19, 0.43, 0.3, 0.77, 0.04 \rangle \tag{2.2}$$

The oracle-trained error is the difference between the target vector in 2.1 and the output in 2.2:

$$\boldsymbol{o} - \hat{\boldsymbol{o}} = \langle 0.08, -0.09, 0.15, 0.12, 0.25 \rangle \tag{2.3}$$

In effect, using the oracle ANN's outputs as targets for the OTN makes the OTN a real-valued function approximator learning to behave like its oracle.

The size of the OTN is chosen according to the given resources. If a given application calls for ANNs no larger than 32 hidden nodes, then a 32 hidden node OTN is created. If there is room for a 2048 hidden node network, then 2048 hidden

nodes is preferable. If the oracle itself meets the resource constraints, then, of course, it should be used in place of an OTN.

2.3.4 Oracle Learning Compared to Semi-supervised Learning

As explained in section 2.2, the idea of labeling unlabeled data with a classifier trained on labeled data is not new [4, 56, 55, 29, 3]. These semi-supervised learning methods differ from oracle learning in that:

1. The goal of semi-supervised learning is to infer novel concepts from unlabeled data, and thus outperform hypotheses obtained using only labeled data. The goal of oracle learning for ANN reduction, however, is only to produce a smaller ANN that learns concepts *already* inferred by the oracle from *only* the labeled data. The unlabeled data merely provides more examples drawn from the same distribution to help the OTN fit its oracle. Theoretically, semi-supervised methods can be used to create an oracle, since the goal in obtaining the oracle is to have the most accurate available model. Once created, oracle learning can then be applied to reduce the size of the oracle created through semi-supervised learning.

2. With oracle learning the data is relabeled with the oracle's exact outputs, not just the class. Preliminary experimentation showed using the exact outputs yielded improved results over using only the class information. We conjecture in section 2.5.2 that the improvement in accuracy comes because using the exact outputs creates a function solving the same classification problem that is easier to approximate using the backpropagation algorithm. Caruana proves this is possible in his work on *RankProp* [9, 10, 11].

Semi-supervised learning could be used as an oracle selection mechanism since it seeks to increase accuracy, but it not does not compare directly with oracle learning as used for model size reduction.

2.4 Methods

The following experiments serve to validate the effectiveness of oracle learning, demonstrating the conditions under which oracle learning best accomplishes its goals. Trends based on increasing the relative amount of oracle-labeled data are shown by repeating each experiment using smaller amounts of hand-labeled data while keeping the amount of unlabeled data constant. Further experiments to determine the effects of removing or adding to the unlabeled data set while keeping the amount of hand-labeled data constant will be conducted as subsequent research because of the amount of time required to do them.

2.4.1 The Applications

One of the most popular applications for smaller computing devices (e.g. hand-held organizers, cellular phones, etc.) and other embedded devices is automated *speech recognition* ASR. Since the interfaces are limited in smaller devices, being able to recognize speech allows the user to more efficiently enter data. Given the demand for and usefulness of speech recognition in systems lacking in memory and processing power, there is a need for smaller ASR ANNs capable of achieving acceptable accuracy.

One of the problems with the ASR application is an element of indirection added by using a phoneme-to-word decoder to determine accuracy. It is possible that oracle learning does well on problems with a decoder and struggles on those without. Therefore, a non-decoded experiment is also conducted. A popular application for ANNs is *optical character recognition* (OCR) where ANNs are used to convert images of typed or handwritten characters into electronic text. Although unlabeled data is not as difficult to obtain for OCR, it is a complex, real-world problem, and therefore good for validating oracle learning. It is also good for proving oracle learning's potential because no decoder is used.

2.4.2 The Data

The ASR experiment uses data from the unlabeled TIDIGITS corpus [45] for testing the ability of the oracle ANN to create accurate phoneme level labels for the OTN. The inputs are the first 13 Mel cepstral coefficients and their derivatives in 16 ms intervals extracted in 10 ms overlapping frames taken at 5 selected time intervals for a total of 130 inputs. The TIDIGITS corpus is partitioned into a training set of 15,322 utterances (around 2,700,000 phonemes), a validation set of 1,000 utterances, and a test set of 1,000 utterances (both 180,299 phonemes). Four subsets of the training corpus consisting of 150 utterances (26,000 phonemes), 500 utterances (87,500 phonemes), 1,000 utterances (175,000 phonemes), and 4,000 utterances (700,000 phonemes) are *bootstrap*-labeled at the phoneme level and used for training the oracle ANN. Only a small amount of speech data has been phonetically labeled because it is inaccurate and expensive. Bootstrapping involves using a trained ANN to force align phoneme boundaries given word labels to create $0-1$ target vectors at the phoneme level. Forced alignment is the process of automatically assigning where the phonemes begin and end using the known word labels and a trained ANN to estimate where the phonemes break and what the phonemes are. Although the bootstrapped phoneme labels are only an approximation, oracle learning succeeds as long as it can effectively reproduce that approximation in the OTNs. Each experiment is repeated using each of the above subsets as the only available labeled data in order to determine how varying amounts of unlabeled data affect the performance of OTNs.

The OCR data set consists of 500,000 alphanumeric character samples partitioned into a 400,000 character training set, a 50,000 character validation set, and a 50,000 character test set. Each data point consists of 64 features from an $8x8$ grid of the gray-scale values of the character. The four separate training set sizes include one using all of the training data (400,000 out of the 500,000 sample set), another using 25% of the training data (100,000 points), the third using 12.5% of the data

(50,000 points), and the last using only 5% of the training data (4,000 points) yielding cases where the OTN sees 20, 8, and 4 times more data than the standard trained ANNs, and case where they both see the same amount of data. In every case the 400,000-sample training set is used to train the OTNs.

2.4.3 Obtaining the Oracles

For each training set size for both ASR and OCR, ANNs of an increasing number of hidden nodes are trained on the labeled training data available for the corresponding size and tested on the corresponding validation set. For example, for the 100,000 point training set of the OCR data, the oracle is trained on only 100,000 points, and tested on the 50,000 point validation set. The size of the oracle ANN is chosen as the ANN with the highest average and least varying accuracy on the validation set (averaged over five different random initial weight settings). The oracle selection process is repeated for each training set, resulting in an oracle chosen for each of the four training set sizes. As an example, figure 2.2 shows mean accuracy and standard deviation for a given number of hidden nodes on a validation set for ANNs trained on the 4,000-utterance ASR training set. The ideal oracle ANN size in this case has 800 hidden nodes since it has the highest mean and lowest standard deviation. The same method is used to choose four oracles for ASR and four for OCR, one for each training set size.

The same decaying learning rate is used to train every ANN (including the OTNs) and starts at 0.025, decaying according to $\frac{.025}{1+\frac{p}{5N}}$ where p is the total number of patterns seen so far and N is the number of patterns in the training set. This has the effect of decaying the learning rate by $\frac{1}{2}$ after five epochs, $\frac{1}{3}$ after 10, $\frac{1}{4}$ after 15, etc. The initial learning rate and its rate of decay are chosen for their favorable performance in our past experiments.

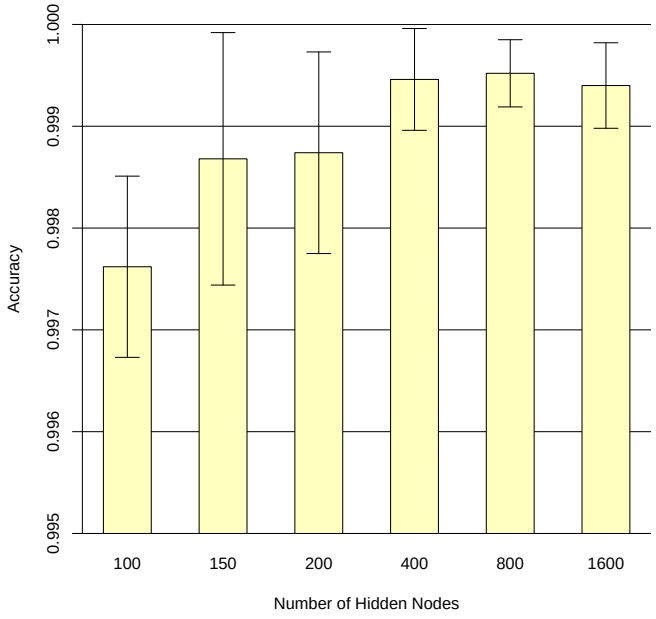

Figure 2.2: Mean accuracy and ± two standard deviations for $100-1600$ hidden node ANNs on the 4,000-utterance ASR training set.

2.4.4 Labeling the Data Set and Training the OTNs

The four ASR oracles from section 2.4.3 create four oracle-labeled training sets using the entire $15,000+$ utterance (2.7 million phoneme) set whereas the four OCR oracles create four training sets using the 400,000 character training set, as described in section 2.3.2. Therefore, the OTNs train on all of the data—$15,000+$ utterances or 400,000 characters—whereas the oracle ANNs and the standard-trained ANNs train on only that fraction of the data chosen for the corresponding training set size. For example, in the 100,000 character case, the oracle ANN and the standard-trained ANN are both trained on the same amount of data—100,000 characters—whereas the OTN is trained on the same 100,000 characters relabeled by its oracle plus 300,000 additional originally unlabeled characters also labeled by the corresponding oracle. The training sets are then used to train ANNs of sizes beginning with the first major break in accuracy, including 100, 50 and 20 hidden node ANNs for ASR and starting

at either 512 or 256 hidden nodes for OCR and decreasing by halves until 32 hidden nodes.

2.4.5 Performance Criteria

For every training set size in section 2.4.2, and for every OTN size, five separate OTNs are trained using the training sets described in section 2.4.4, each with different, random initial weights. Even though there are only five experiments per OTN size on each training set, for ASR there are a total of 20 experiments for each of the three sizes across the four training sets and 15 experiments for each of the four training sets across the three sizes (for a total of $20 \cdot 3$ or $15 \cdot 4 = 60$ experiments). For OCR, this yields 10-20 experiments for each of the four OTN sizes and 20-25 experiments per training set size for at total of 90 experiments. After every oracle learning epoch, recognition accuracies are gathered using a hold-out set. The ANN most accurate on the hold-out set is then tested on the test set. The five test set results from the five OTNs performing best on the hold-out set are then averaged for the reported performance. The accuracies were measured out to 4 significant digits because in some cases the interesting changes were out at that level. The resulting accuracies are compared to ANNs trained using standard methods. The standard trained ANNs are trained on the same data as the oracles for each training set size.

2.5 Results and Analysis

2.5.1 Results

Table 2.1 summarizes the results of oracle learning. To explain how the table reads, consider the first row. The set size column has ASR and 150 meaning this is part of the ASR experiment and this row's results were obtained using the 150 utterance training set. The OTN size column states 20 meaning this row is comparing 20

Table 2.1: Combined ASR and OCR Results showing % decrease in error compared to standard training and oracle similarity.

Set Size	OTN Size	% Dec. Error vs. Standard	% Avg Dec. error for OTN Size	Similarity	Average Similarity for OTN Size
ASR	20	36.01	21.47	0.9934	0.9916
150	50	22.96	16.44	0.9981	0.9980
	100	9.61	7.56	0.9998	0.9994
	Avg	22.86		0.9971	
	20	34.92	21.47	0.9937	0.9916
500	50	26.77	16.44	0.9994	0.9980
	100	20.00	7.56	1.0001	0.9994
	Avg	27.23		0.9977	
	20	2.54	21.47	0.9895	0.9916
1,000	50	15.09	16.44	0.9977	0.9980
	100	-5.29	7.56	0.9989	0.9994
	Avg	4.11		0.9953	
	20	12.42	21.47	0.9900	0.9916
4,000	50	0.95	16.44	0.9967	0.9980
	100	5.94	7.56	0.9989	0.9994
	Avg	6.44		0.9953	
Avg		**15.16**		**0.9964**	

Set Size	OTN Size	% Dec. Error vs. Standard	% Avg Dec. error for OTN Size	Similarity	Average Similarity for OTN Size
OCR	32	29.71	17.41	0.9773	0.9688
5 %	64	24.64	17.67	0.9931	0.9889
	128	12.21	11.05	0.9977	0.9962
	256	4.46	4.04	0.9994	0.9989
	Avg	17.75		0.9919	
	32	23.09	17.41	0.9704	0.9688
12.5%	64	24.16	17.67	0.9906	0.9889
	128	15.61	11.05	0.9970	0.9962
	256	4.98	4.04	0.9992	0.9989
	512	2.18	2.24	0.9999	0.9999
	Avg	14.00		0.9914	
	32	14.93	17.41	0.9674	0.9688
25%	64	18.46	17.67	0.9883	0.9889
	128	13.06	11.05	0.9962	0.9962
	256	5.59	4.04	0.9993	0.9989
	512	2.30	2.24	1.0000	0.9999
	Avg	10.87		0.9902	
	32	1.91	17.41	0.9600	0.9688
100%	64	3.43	17.67	0.9836	0.9889
	128	3.31	11.05	0.9939	0.9962
	256	1.14	4.04	0.9977	0.9989
	Avg	2.45		0.9838	
Avg		**11.40**		**0.9895**	

hidden node OTNs to 20 hidden node standard ANNs. The % Dec. Error column reads 36.01 meaning that using oracle learning resulted in a 36.01% decrease in error over 20 hidden node ANNs trained using standard methods on the same data used to train the oracle (i.e. the 150 utterance training set). Decrease in error is calculated as:

$$1 - \frac{Error_{OTN}}{Error_{Standard}} \tag{2.4}$$

The % Dec. error for OTN Size column gives 21.47 meaning averaged over the four training set sizes, oracle learning results in a 21.47% decrease in error over standard training for ANNs with 20 hidden nodes. This number is repeated for each training set size for convenience in reading the table. The *similarity* column reads 0.9934 meaning that 20 hidden node OTNs retain 99.34% of their oracle's accuracy for the 150 utterance case. Similarity is calculated as:

$$Similarity = \frac{Accuracy_{OTN}}{Accuracy_{Oracle}} \tag{2.5}$$

The average similarity for OTN size column reads 0.9916 meaning that averaged over the four training set sizes, 20 hidden node networks retain 99.16% of their oracles' accuracy. The average decrease in error using oracle learning instead of standard methods is 15.16% averaged over the 60 experiments for ASR and 11.40% averaged over the 90 experiments for OCR. OTNs retain 99.64% of their oracles' accuracy averaged across the 60 experiments for ASR and 98.95% of their oracles' accuracy averaged across the 90 experiments for OCR.

The results give evidence that as the amount of available labeled data decreases without a change in the amount of oracle-labeled data, oracle learning yields more and more improvement over standard training. This is probably due to the OTNs always having the same large amount of data to train on. They experience far more data points, and even though they are labeled by an oracle instead of by hand, the

quality of the labeling is sufficient to exceed the accuracy attainable through training on only the hand labeled data.

2.5.2 Analysis

The results above provide evidence that oracle learning can be beneficial when applied to either ASR or OCR. With only one exception, the OTNs have less error than their standard trained counterparts. Oracle learning's performance improves with respect to standard training if either the amount of labeled data or OTN size decreases. Therefore, for a given ASR application with only a small amount of labeled data, or given a case where an ANN of 50 hidden nodes or smaller is required for ASR or an ANN of 128 hidden nodes or less is required for OCR, oracle learning is particularly appropriate. The ASR 20 hidden node OTNs are an order of magnitude smaller than their oracles and are able to maintain 99.16% of their oracles' accuracy averaged over the training set sizes with 21.47% less error than standard training. The 32 hidden node OTNs are two orders of magnitude smaller than their oracles but maintain 96.74% of their oracles' accuracy with 17.41% less error than standard training. On average, oracle learning results in a 15.16% decrease in error for ASR and an 11.40% decrease in error for OCR compared to standard methods. Oracle learning also allows the smaller ANNs to retain 99.64% of their oracles' accuracy on average for ASR and 98.95% of their oracles' accuracy on average for OCR.

Oracle learning's positive performance is mostly likely due to there being enough oracle-labeled data points for the OTNs to effectively approximate their oracles. Since the larger, standard-trained oracles are always better than the smaller, standard-trained ANNs, OTNs that behave *like* their oracles are more accurate than their standard-trained equivalents. Another reason for oracle learning's performance may be that since the OTNs train to output targets between 0 and 1 instead of exactly 0 and 1, oracle learning presents a function that is easier for backpropagation to

learn. Caruana [9, 10, 11] proved that for any classification function $f(x)$, there may exist a function $g(x)$ that represents the same solution, but is easier for backpropagation to learn. If this is the case for oracle learning, future work may result in a novel algorithm that produces easier, but equivalent functions for backpropagation to learn. However, future work will first focus on determining how varying the amount of available unlabeled instead of labeled data affects the performance of oracle learning.

2.5.3 Oracle Learning Compared to Pruning

As stated in section 2.2, both oracle learning and pruning can be used to produce a smaller ANN. The main difference is that with oracle learning, the smaller model is created initially and trained using the larger model, whereas with pruning, connections are removed from the larger model until the desired size is reached.

Autoprune [25] is selected to compare pruning to oracle learning because it is shown to be more successful [59, 25] than the more popular methods in [63], and because it has a set pruning schedule. *Lprune* [59] adds an adaptive pruning schedule to autoprune, but it is designed to improve generalization accuracy and not to reduce model size explicitly. Lprune decides automatically how many connections to prune in terms of validation set accuracy, and may therefore never yield the desired number of connections. Lprune is meant to be used to improve overall accuracy and not, as is the case of oracle learning, to reduce the size of ANNs.

To compare autoprune to oracle learning, first a 128 hidden node ANN is trained. Next, the 128 hidden node ANN is used as the oracle to train a 32 hidden node OTN (see section 2.3). Then, the larger ANN is pruned using autoprune's schedule of first pruning 35% of the weights, then retraining until, in this case, a hold-out set suggests early stopping. The pruned ANN is then allowed to train once again until performance on a hold-out set stops increasing. After the pruned ANN is retrained, 10% of the remaining connections are pruned, followed by more retraining.

Table 2.2: Oracle learning compared to autoprune

Model	# Connections	Relative Size	Epochs	Accuracy
Original ANN	18,944	4.00	33	0.9656
Auto-Pruned	8,079	1.71	1634	0.9415
OTN	4,736	1.00	198	0.9338
STD	4,736	1.00	256	0.9266
Auto-Pruned	7,271	1.54	2201	0.9199
Auto-Pruned	4,771	1.01	3774	0.6113
Auto-Pruned	4,294	0.91	4188	0.4966

This process continues pruning 10% at a time with results reported in two places. First, where the error of the pruned ANN is similar to that of the oracle trained ANN, and second, when the number of connections of the pruned ANN is similar to that of the oracle trained ANN. Table 2.2 shows the results of the experiment in order of highest accuracy. The top model is the original 128 hidden node ANN also used as the oracle ANN. The second line shows the auto-pruned ANN with an accuracy just above that of the OTN, the next model is the oracle trained ANN, the fourth compares the results of training a 32 hidden node network using standard methods, the fifth result shows the auto-pruned ANN with an accuracy just below that of the OTN, and the bottom two results show the auto-pruned ANNs with a similar number of connections to the OTN. Notice that the pruned ANNs need more connections (1.54 to 1.71 times more) to obtain similar results to the OTN. Notice also that when allowed to prune to the same number of connections as the OTN, the auto-pruned ANNs have significantly lower accuracy. The results show pruning is not as effective as oracle learning when reducing an initial model to a specified size. In addition, the pruned ANNs required 10-20 times more training epochs. This suggests pruning is also less efficient than oracle learning.

2.5.4 Bestnets

As mentioned in section 2.1, another interesting area where we have successfully applied oracle learning is approximating a set of ANNs that are "experts" over specific parts of a given function's domain. The oracle learning solution we propose, namely *bestnets*, uses the set of "experts" as oracles to train a single ANN to approximate their performance.

The application in which we have successfully applied the bestnets method is to improve accuracy when training on data with varying levels of noise. A common solution to learning in a noisy environment is to train a single classifier on a mixed data set of both clean and noisy data. Often, the resulting classifier performs worse on clean data than a classifier trained only on clean data, and likewise for a classifier trained only on noisy data. It would be preferable to use the two domain specific classifiers instead, but this requires knowing if a given sample is clean or noisy—an often difficult problem to solve. Here, oracle learning can be used to approximate the two domain specific classifiers with a single OTN. The classifier trained only on noisy data and the classifier trained only on clean data can be used together as an oracle to label those parts of the training set that correspond to each model's "expertise."

Initial results from McNemar tests [19] are shown in table 2.3. The ANN1 and ANN2 columns give the type of data used to train the models being compared. The data set column shows the type of data used to compare the two ANNs. The difference column gives the accuracy of ANN1 minus the accuracy of ANN2. The p-value column gives the p-value (lower is better) resulting from a McNemar test [19] for statistical difference between ANN1 and ANN2. A p-value of less than 0.0001 means that a difference as extreme or more than the difference between the two ANNs compared would be seen randomly less than 1 out of 10000 repeats of the experiment. In other words, the test is more than 99% confident that ANN1 is better than ANN2. Notice in the first two rows that the ANN trained on mixed data is

Table 2.3: Results comparing the bestnets method to training directly on mixed data.

ANN1	ANN2	Data set	Difference	p-value
Clean Data Oracle	Mixed Data	Clean	0.0307	< 0.0001
Noisy Data Oracle	Mixed Data	Noisy	0.0092	< 0.0001
Bestnets	Mixed Data	Mixed	0.0056	< 0.0001
Clean Data Oracle	Bestnets	Clean	0.0298	< 0.0001
Noisy Data Oracle	Bestnets	Noisy	-0.0011	0.1607

significantly worse than the "expert" ANNs trained on their specific domains. This suggests there is room for improvement by using the bestnets method. The third row in the table shows that the bestnets ANN is significantly better than the ANN trained on mixed data when compared using both the noisy and clean data. It is expected that the bestnets ANN will miss one less in every 250 characters than the mixed data trained-ANN. The bestnets ANN is not significantly different than the ANN trained only on noisy data, yielding another improvement over directly training on the mixed data set. The bestnets method improves over training on the mixed data and retains the performance of the noise specific ANN. Future work will focus on increasing the improvement in accuracy of the bestnets method over standard training, especially on the clean data.

2.6 Conclusion and Future Work

2.6.1 Conclusion

On automatic spoken digit recognition oracle learning decreases the average error by 15.16% over standard training methods while still maintaining, on average, 99.64% of the oracles' accuracy. For optical character recognition, oracle learning results in a 11.40% decrease in error over standard methods, maintaining 98.95% of the oracles' accuracy, on average. The results also suggest oracle learning works best under the following conditions:

1. The size of the OTNs is small.

2. The amount of available hand-labeled data is small.

2.6.2 Future Work

One area of future work will investigate the effect of varying the amount of available unlabeled instead of labeled data. Another will determine if oracle learning presents a function that is easier to learn for back-propagation than the commonly used 0-1 encoding. If oracle learning does create an easier function, future research may lead to novel learning mechanisms that produce equivalent, but easier functions for back-propagation to learn, therefore outperforming standard training on any data set, regardless of the availability of unlabeled data. A third area of future work will focus on increasing the bestnets method's gains over direct training in a environment with varying levels of noise.

Chapter 3

Domain Expert Approximation Through Oracle Learning

Abstract

In theory, improved generalization accuracy can be obtained by training separate learning models as "experts" over subparts of a given application domain. For example, given an application with both clean and noisy data, one solution is to train a single classifier on a set of both clean and noisy data. More accurate results can be obtained by training separate expert classifiers, one for clean data and one for noisy data, and then using the appropriate classifier depedning on the environment. Unfortunately, it is usually difficult to distinguish between clean and noisy data outside of training. We present a novel approach using *oracle learning* to approximate the clean and noisy domain experts with one learning model. On a set of both noisy and clean optical character recognition data, using oracle learning to approximate domain experts resulted in a statistically significant improvement ($p < 0.0001$) over using a single classifier trained on mixed data.

3.1 Introduction

The main idea in *oracle learning* [51, 48] is that instead of training directly on a set of data, a learning model is trained to approximate a given *oracle's* behavior on a set of data. The oracle can be another learning model that has already been trained on the data, or it can be any given functional mapping $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ where n is the number of inputs to both the mapping and the *oracle-trained model* (OTM), and m is the number of outputs from both. The main difference with oracle learning is that the OTM trains on a training set whose targets have been relabeled by the oracle instead of training with the original training set labeling. Having an oracle to label data means that previously unlabeled data can also be used to augment the relabeled training set. The key to oracle learning's success is that it attempts to use a training set that fits the observed distribution of the given problem to accurately approximate the oracle on those sections of the input space that are most relevant in real-world situations. In [51, 48] small *artificial neural networks* (ANNs) are trained to approximate larger ANNs instead of being trained directly on the data. In addition, the smaller ANNs are trained on previously unlabeled data since the larger ANNs can serve as *oracle ANNs* to label data that did not originally have labels. In the following, instead of approximating a single ANN, we use a set of *domain expert* ANNs as an oracle to train a single ANN on real data.

For a given application, higher generalization accuracy can be obtained by training separate learning models as "experts" over specific domains. For example, given an application where it is common to observe at least two varying levels of noise, clean and noisy, one solution is to train a single classifier on both clean and noisy data. It is possible to achieve better accuracy by training one classifier on only noisy data and one classifier on only clean data, and then choosing between them during classification depending on the environment. The clean and noisy domain experts will have higher accuracy on their respective domains than a classifier trained on a mix of

both clean and noisy data. Unfortunately, it is difficult to know beforehand whether a given data point belongs to the clean or noisy section of the data, and therefore it is difficult to know whether to use the clean or noisy domain expert. Here, we present the *bestnets* method which uses *oracle learning* to approximate the behavior of both the clean and noisy domain experts with a single learning model.

In [51, 48], oracle learning is used to approximate a single, larger ANN. With the bestnets method, oracle learning is used to approximate the behavior of multiple ANNs, each expert on parts of a given application's domain. When given a problem that has both noisy and clean data, one domain expert ANN is trained only on clean data, and another expert is trained only on noisy data. Then, each domain expert relabels the original training data on that expert's part of the domain. Furthermore, because the domain experts can be used to label any given data point, the original training data can be augmented by previously unlabeled data, creating an even larger training set. Note that this unlabeled data is only used to better approximate the domain experts, not for inferring additional concepts beyond what the domain experts learned from the original training set. Finally, a single ANN, the *bestnets-ANN*, is trained on the domain expert-labeled training set. On a set of both noisy and clean optical character recognition data, using oracle learning to approximate the domain experts resulted in a statistically significant improvement ($p < 0.0001$) over standard training on the mixed data.

3.2 Background

The idea of approximating a model is not new. [20] used Quinlan's C4.5 decision tree approach [61] to approximate a bagging ensemble. [8] and [78] used an ANN to approximate a similar ensemble [8]. Craven and Shavlik used a similar approximating method to extract rules [14] and trees [15] from ANNs. Domingos [20] and Craven and Shavlik [14, 15] used their ensembles to generate training data where the

targets were represented as either being the correct class or not. Zeng and Martinez [78] used a target vector containing the exact probabilities output by the ensemble for each class. The following research also uses vectored targets similar to Zeng and Martinez since Zeng's results support the hypothesis that vectored targets "capture richer information about the decision making process ..." [78]. Menke et. al used *oracle learning* in [51, 48] to reduce the size of ANNs by approximating larger ANNs with smaller ANNs, using unlabeled data. While previous research has focused on either extracting information from ANNs [14, 15], using statistically generated data for training [20, 78], or reducing the size of ANNs [51, 48], the novel approach presented here is that a single ANN can be trained using oracle learning to approximate multiple domain experts.

3.3 Bestnets

There are three major steps in the bestnets learning process and the following sections describes each one in detail. First, the domain experts need to be trained correctly. Then, the domain experts are used to relabel the original training data changing the targets to the exact outputs of the correct domain expert on each data point. Finally, the bestnets-ANN is trained using the relabeled dataset.

3.3.1 Obtaining the Domain Experts

Since the accuracy of the domain experts directly influences the performance of the bestnets-ANN, the domain experts must be the most accurate classifiers available for their domains, regardless of complexity (number of hidden nodes). In the case of ANNs, the most accurate classifier is usually the largest ANN that improves over the next smallest ANN on a validation set. The domain experts should be chosen using a validation (hold-out) set (or similar method) in order to prevent over-fitting the data. For our domain experts, we choose the most accurate ANN improving over the

next smallest ANN on a validation or hold-out set in order to prevent the larger ANN from over-fitting.

3.3.2 Labeling the Data

The key to the bestnets method is being able to use knowledge of the domain at train-time to augment later generalization. Since the training set contains information indicating which data is clean and which noisy, that knowledge can be incorporated into a single classifier using the bestnets method allowing the bestnets-ANN to implicitly distinguish between clean and noisy data and "mimic" the behavior of an expert over that domain. This is accomplished by training an ANN to give the same outputs as the "clean oracle" when given clean data, and likewise give the same outputs a "noisy oracle" would give when presented with noisy data.

To train an ANN to behave in this manner, the clean data used to originally train the clean domain expert is relabeled by the clean domain expert with the exact outputs of the clean domain expert on each training point. The same is done with the noisy domain expert. In other words, this step creates a target vector $\mathbf{t^j} = \mathbf{t_1} \ldots \mathbf{t_n}$ for each input vector $\boldsymbol{x}^j$ from a given domain where each t_i is equal to the domain expert's activation of output i given the j^{th} pattern in the data set, $\boldsymbol{x}^j$. Then, the final bestnets data point contains both $\boldsymbol{x}^j$ and $\boldsymbol{t}^j$. In order to create the labeled training points, each available pattern $\boldsymbol{x}^j$ is presented as a pattern to its respective domain expert which then returns the output vector $\boldsymbol{t}^j$. The final bestnets training set then consists of the pairs $\boldsymbol{x}^1\boldsymbol{t}^1 \ldots \boldsymbol{x}^m\boldsymbol{t}^m$ for all m data points—both clean and noisy.

3.3.3 Training the Bestnets ANN

For the final step, the bestnets-ANN is trained using the data generated in section 3.3.2, utilizing the targets exactly as presented in the target vector. The bestnets-

ANN interprets each real-valued element of the target vector $\boldsymbol{t}^j$ as the correct output activation for the output node it represents given $\boldsymbol{x}^j$. The back-propagated error is therefore $t_i - o_i$ where t_i is the i^{th} element of the target vector $\boldsymbol{t}^j$ (and also the i^{th} output of the domain expert) and o_i is the output of node i. This error signal causes the outputs of the bestnets-ANN to approach the target vectors of the domain expert corresponding to each data point as training continues.

As an example, the following vector represents the output vector $\boldsymbol{o}$ of the noisy domain expert given the input vector $\boldsymbol{x}$ from a set of noisy data. $\hat{\boldsymbol{o}}$ represents the output of the bestnets-ANN. Notice the 4^{th} output is the highest and therefore the correct one as far as the domain expert is concerned.

$$\boldsymbol{o} = \langle 0.27, 0.34, 0.45, 0.89, 0.29 \rangle \tag{3.1}$$

Now suppose the bestnets-ANN outputs the following vector:

$$\hat{\boldsymbol{o}} = \langle 0.19, 0.43, 0.3, 0.77, 0.04 \rangle \tag{3.2}$$

The error is the difference between the target vector in 3.1 and the output in 3.2:

$$\boldsymbol{o} - \hat{\boldsymbol{o}} = \langle 0.08, -0.09, 0.15, 0.12, 0.25 \rangle \tag{3.3}$$

In effect, using the domain expert's outputs as targets for the bestnets-ANN makes the bestnets-ANN a real-valued function approximator learning to behave like the appropriate domain expert on each domain.

3.4 Experiment and Results

In order to test the effectiveness of bestnets training, an experiment was conducted using *optical character recognition* (OCR) data containing both noisy and clean sam-

ples. The clean OCR data set consists of 500,000 alphanumeric character samples randomly partitioned into a 400,000 character training set, a 50,000 character validation set, and a 50,000 character test set. Each data point consists of 64 features from an $8x8$ grid of the gray-scale values of the character. The noisy OCR data set was created from the clean set by adding a random amount of noise to each of the 64 pixels in the $8x8$ grid. A different amount of noise was chosen for each pixel by randomly generating a number from a standard normal distribution, cubing it, dividing it by 3, and then adding it to the pixel's original value. Pixel values were then clipped to $[0, 1]$. This is repeated for each pixel on each pattern creating the "salt and pepper" effect seen in figure 3.1.

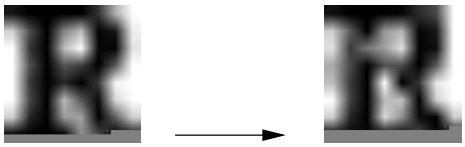

Figure 3.1: The character R before and after adding random noise

The bestnets learning method as described in section 3.3 was applied to the OCR set. Both clean and noisy domain experts were created by training ANNs on clean-only and noisy-only data respectively. The domain experts were then used to relabel the original data, and then the domain expert-labeled data was used to train the bestnets-ANN. Results were obtained and compared to the domain experts and to training on mixed data without domain expert labels.

Results from McNemar tests [19] are shown in table 3.1. The ANN1 and ANN2 columns give the type of data used to train the models being compared. The data set column shows the type of data used to compare the two ANNs. The difference column gives the accuracy of ANN1 minus the accuracy of ANN2. The p-value column gives the p-value (lower is better) resulting from a McNemar test [19] for statistical difference between ANN1 and ANN2. A p-value of less than 0.0001 means that a difference as extreme or more than the difference between the two ANNs compared

would be seen randomly less than 1 out of 10000 repeats of the experiment. In other words, the test is more than 99% confident that ANN1 is better than ANN2. Notice in the first two rows that the ANN trained on mixed data is significantly worse than the "expert" ANNs trained on their specific domains. This suggests there is room for improvement by using the bestnets method. The third row shows that the bestnets model was still significantly worse than the clean domain expert, and therefore there is still room for improvement on at least one of the domains. The fourth row in the table shows that the bestnets-ANN is not significantly different than the ANN trained only on noisy data, yielding an improvement over directly training on the mixed data set and showing the bestnets method retaining the performance of one of the domain experts. Finally, the last row shows that the bestnets-ANN is significantly better than the ANN trained on mixed data when compared using both the noisy and clean data. It is expected that the bestnets-ANN will miss one less in every 250 characters than the mixed data trained-ANN. This final row is the most interesting because it compares two solutions to the problem that are realistic since the domain experts can not be used directly without a way of distinguishing explictly whether a given data point is clean or noisy.

ANN1	ANN2	Data set	Difference	p-value
Clean Data Oracle	Mixed Data	Clean	0.0307	< 0.0001
Noisy Data Oracle	Mixed Data	Noisy	0.0092	< 0.0001
Clean Data Oracle	Bestnets	Clean	0.0298	< 0.0001
Noisy Data Oracle	Bestnets	Noisy	-0.0011	0.1607
Bestnets	**Mixed Data**	**Mixed**	**0.0056**	**$<$0.0001**

Table 3.1: Results comparing the bestnets method to training directly on mixed data.

3.5 Conclusions

The bestnets method improves over training on the mixed data and retains the performance of the noisy domain expert. Future work will investigate why there was not

as significant of an improvement on the clean data. In order to determine where the bestnets method's ability to retain domain expert accuracy diminishes, experiments with several varying levels of noise will be conducted and then the bestnets method will be modified to preserve the experts' accuracy where there is currently room for improvement.

The bestnets method can be used for more than just approximating experts on varying levels of noise. One area of future work will develop methods to automatically identify subsections in a given application domain. Then, the bestnets method can be applied over the subsections just as applied here on varying levels of noise. In this manner, bestnets can be applied to a wide range of problems, not just those for which divisions are known beforehand.

Chapter 4

Improving Machine Learning By Adapting the Problem to the Learner

Abstract

While no machine learning algorithm can do well over all functions, we show that it may be possible to adapt a given function to a given machine learning algorithm so as to allow the learning algorithm to better classify the original function. Although this seems counterintuitive, adapting the problem to the learner may result in an equivalent function that is "easier" for the algorithm to learn. The following presents two problem adaptation methods, SOL-CTR-E and SOL-CTR-P, variants of *Self-Oracle Learning with Confidence-based Target Relabeling* (SOL-CTR) as a proof of concept for problem adaptation. The SOL-CTR methods produce "easier" target functions for training *artificial neural networks* (ANNs). Applying SOL-CTR over 41 data sets consistently results in a statistically significant ($p < 0.05$) improvement in accuracy over 0/1 targets on data sets containing over 10,000 training examples.

4.1 Introduction

It is well known that no machine learning algorithm does well over all functions [74], however it may be possible to adapt a given function to better fit a given learning algorithm. Instead of only training the learner on the problem, we show that the problem can be "trained" on the learner simultaneously, in order to improve performance. Adapting the problem to the learner may result in an equivalent function that is "easier" for a given algorithm to learn. As a special case example, consider *rankprop* [11]. Caruana showed that given a standard classification function $f(x)$ with 0/1 targets, and a problem where the goal is learning to sort the patterns instead of directly modeling $f(x)$, there can exist a function $g(x)$ that models the sorting of the patterns by $f(x)$. Caruana showed that $g(x)$ can be "easier to learn" for backpropagation-trained *artificial neural networks* (ANNs) and was able to obtain higher accuracy training on $g(x)$ than training on $f(x)$. Rankprop is designed for single-output problems where ranking is appropriate (e.g. ranking patient priorities for admittance to the hospital). It would be desirable to develop a learning method that adapted any problem to its learner without having specific restraints like needing to sort the data in some fashion. We propose a more general approach which takes an arbitrary data set and modifies that data set to better fit the learning algorithm such that the learning algorithm attains higher classification accuracy on a test set taken from the original data set. It is a method for target relabeling that combines *self-oracle learning* (SOL), based on a training paradigm called *oracle learning* [52], and ANN confidence measures. SOL is a proof of concept method to demonstrate the potential for adapting problems to the learner.

The main idea in oracle learning is that instead of training directly on a set of data, a learning model is trained to approximate a given *oracle's* behavior on a set of data. The oracle can be another learning model that has already been trained on the data, or it can be any given functional mapping $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ where n is the

number of inputs to both the mapping and the *oracle-trained model* (OTM), and m is the number of outputs from both. The main difference with oracle learning is that the OTM trains on a training set whose targets have been relabeled by the oracle instead of training with the original training set labeling. Having an oracle to label data means that previously unlabeled data can also be used to augment the relabeled training set. The key to oracle learning's success is that it attempts to use a training set that fits the observed distribution of the given problem to accurately approximate the oracle on those sections of the input space that are most relevant in real-world situations. In [52] small ANNs are trained to approximate larger ANNs instead of being trained directly on the data. In addition, the smaller ANNs are trained on previously unlabeled data since the larger ANNs can serve as *oracle ANNs* to label data that did not originally have labels. Using oracle learning to reduce the size of these ANNs resulted in a 15% decrease in error over standard training and maintained a significant portion of the oracles' accuracy while being as small as 6% of the oracles' size. [50] also use oracle learning in the *bestnets* algorithm to approximate multiple domain experts with a single ANN.

Although oracle learning allows for the use of unlabeled data to augment existing training sets, [52] showed that even when no unlabeled data was used, it was possible in some cases to achieve higher accuracy using the oracle-labeled targets instead of the original 0/1 encoding. The higher accuracy suggests oracle learning may be creating an "easier function" for backpropagation to learn, but without requiring a specific meaning to the encoding as is the case with rankprop. The following paper uses the same principle, except that with SOL, an ANN acts as its own oracle to relabel the standard training set. The set is labeled with the ANN's exact outputs on the data points as training progresses, instead of having a separate oracle model for relabeling. SOL is used to determine the difficulty level of each data point. If the

ANN is struggling with certain data points, or has already learned certain points, the final targets can be adapted to reflect the current estimated difficulty of each point.

The problem with using an ANN to relabel the training set with its own outputs is that the ANN can be wrong in its predictions, and using its exact outputs as labels for the data can discard information about the true class of each data point. In order to preserve the correctness of the training set, the original 0/1 output targets are combined with the ANN's outputs using measures of the ANN's confidence in its own outputs. When the ANN is very confident, the labels are more likely to be similar to the ANN's own outputs. When the ANN is less confident, the labels will approach the original 0/1 encoding. Combining SOL with ANN confidence measures yields final targets that are customized for each data point based on the ANN's own measure of the data point's difficulty combined with the ANN's confidence in that measure. Applying *Self-Oracle Learning with Confidence-based Target Relabeling* (SOL-CTR) over 41 data sets consistently results in a statistically significant ($p < 0.05$) improvement in accuracy over 0/1 targets on the data sets containing over 10,000 training examples.

The paper will proceed as follows: section 4.2 gives a background in SOL-CTR, section 4.3 describes SOL-CTR and two of its variants, section 4.4 outlines the experimental methodology used to test SOL-CTR, section 4.5 reviews the results of the experiments, and section 4.6 gives conclusions about SOL-CTR and directions for future work.

4.2 Background

Besides the aforementioned oracle learning and rankprop methods, another area related to SOL is semi-supervised learning, where the model being trained is used to label unlabeled data to improve training accuracy [4]–[3]. The basic approach, often known as semi-supervised learning or boot-strapping, is to train on given labeled data,

classify a different set of unlabeled data, and then train using both the original and the relabeled data. This process is repeated multiple times until generalization accuracy stops increasing. In an oracle learning sense, the model trained is acting as its own oracle, similar to SOL. There are two main differences between semi-supervised learning and SOL. First, semi-supervised learning does not relabel the labeled training set, only the unlabeled data, whereas SOL relabels all the data. Second, semi-supervised learning uses the usual 0/1 encoding, whereas SOL seeks to replace the 0/1 labels and create a function that, like rankprop's, is "easier" for backpropagation-trained ANNs to learn.

Another area of research related to relabeling is Rimer's *classification-based* training algorithm, CB1 [66]. The main idea with the CB1 algorithm is to only backpropagate error when the training ANN misclassifies the current data point or does not have a large enough margin between the correct class and closest incorrect classes' output. Even then, the error is only backpropagated along the outputs that were too high or too low. This is related to SOL because it is another way to produce a potentially simpler and yet equivalent classification function for backpropagation to learn. The main difference with SOL is that it seeks to improve the targets themselves whereas the CB1 algorithm seeks to improve how the error with respect to the targets should be determined. SOL-CTR still backpropagates error on every output, although less for more confident outputs than others, whereas the CB1 algorithm will not backpropagate any error if the ANN is confident enough in its output.

Other areas that are less directly related to SOL, but still worth mentioning include using non-0/1 (or non–1/+1) targets, adaptive learning rate methods, and regularization methods like weight decay and pruning. It is common to suggest using targets other than those at the asymptotes of the transfer function (e.g., using 0.1/0.9 instead of 0/1 with a sigmoid) so that the targets can be reached through training and weights are not needlessly saturated. More formal methods [44] have also been

suggested for choosing exactly where to place the targets given the transfer function. Although using targets other than 0/1 may be another way of creating an "easier" function, SOL-CTR takes this concept to an adaptive level, where the targets are customized for each output based on ANN performance rather than choosing a set of static, non-adaptive targets, that are used the same with every training data point. Adaptive learning rate methods like *rprop* [65], *quickprop* [23], and *conjugate gradient methods* [69] do customize the amount of error backpropagated for a training set at each epoch in training, however the goal is generally faster convergence by taking larger steps along a predicted gradient rather than improving accuracy. SOL-CTR changes the error surface altogether to one that is hopefully easier for backpropagation to converge on, resulting in higher accuracy rather than faster convergence. One reason SOL-CTR may work is because it leads to smaller magnitude weights, and is therefore less likely to overfit [2]. This can be compared to the weight decay [41] regularization method, where weights constantly shrink if they are not being updated. The difference is while weight decay is usually done the same on each weight, SOL-CTR will customize the affect on each weight. In addition, instead of constantly penalizing unused weights, SOL-CTR tries to only use the weights needed at a given point in training. Pruning [63], another form a regularization, will remove unused weights altogether, whereas SOL-CTR will instead try and use the unused weights more efficiently.

Procedure `SOL`(*training set, hold-out set*)

while *hold-out set accuracy increases* **do**
  Initialize ANN to same random weights.
  while *hold-out set accuracy increases* **do**
    Train ANN one epoch on the training set with current labels.
  foreach *data point in the training set* **do**
    Obtain the trained ANN's output on the point.
    Relabel the point's targets with the ANN's current exact outputs.

Figure 4.1: Brute-Force SOL

4.3 Self-Oracle Learning with Confidence-based Target Relabeling

4.3.1 Self-Oracle Learning

As mentioned in section 4.2, SOL uses the ANN being trained as its own oracle in order to find better targets for the training data points. In its simplest form, SOL can be applied as shown in figure 4.1.

Note that the same random weights are used on each iteration of SOL. This is because SOL is designed to learn the correct targets for a given initial weight setting. This ensures that new targets are adapted to not only the structure of the ANN, but also its starting position. When used in this manner, SOL becomes a brute force search for better targets, based on what the ANN is outputting. The idea is that the outputs the ANN is actually able to produce better represent its capacity for fitting the given training data. Applying this approach to a large, noisy OCR data set in preliminary experiments resulted in an ANN that was just as accurate as standard training, but represented a *simpler* function.

Simpler is defined in this case by the final magnitude of the ANN's weights. This measure is appropriate since the bias of backpropagation-trained ANNs is to move from simple to more complex as training progresses. This bias results from the

fact that ANN weights are initialized to small, random values near 0. As the ANN trains, the weights move away from 0 to adapt to the data. A smaller average final magnitude weight implies a simpler function [2]. In the preliminary results, the self-oracle-trained ANN achieved equivalent accuracy to a 0/1 target trained ANN with 71% of the final weight magnitude, yielding a simpler function than the 0/1 targets. Since the resulting ANN is simpler, SOL may be creating a function that is "easier" for backpropagation to learn, resulting in a more efficient use of the weights.

The problem with using SOL as described here is that relabeling the targets exactly as output by the ANN itself means discarding relevant information about the true class of data points that are misclassified. It is possible to achieve higher accuracy by preserving the known class information in the final targets, but the question then becomes how much of the original 0/1 labels to use, and how much of the ANN's outputs should be used. The SOL-CTR approach weights each based on a heuristic used to measure the confidence of the ANN.

4.3.2 ANN Confidence Measures

One method for combining the outputs of the ANN with the original 0/1 labels is to weight each by the ANN's confidence. Thus the new target T for output j of data point i becomes:

$$T_{i,j} = \alpha C_{i,j} O_{i,j} + (1 - \alpha C_{i,j}) S_{i,j} \tag{4.1}$$

where $O_{i,j}$ is the value of the jth output node of the ANN given data point i, $C_{i,j}$ is the ANN's confidence that $O_{i,j}$ is correct, and $S_{i,j}$ is the original 0/1 encoding of the target outputs for data point i. The variable α is a meta-level trust value placed on the confidence measure C. If C is known to be exactly accurate, α can be set to 1. If there is some meta-level uncertainty about the parameter C, then α can be set to reflect that uncertainty. Therefore the output of each data point is trained using a

target customized for that exact output and data point, based on the ANN's output and confidence in that output.

In theory, the quantity $C_{i,j}$ given above can not be measured directly since it will always be 0. This is because $C_{i,j}$ represents the evaluation of a continuous probability density function at a single point. Given a continuous density function, probabilities are measured over intervals, and here the interval and probability are both 0. Therefore, instead of trying to measure this quantity directly, a heuristic is used that measures the ANN's confidence in each class. The heuristic chosen for this paper is the *F-measure*. The F-measure for class k is determined as follows:

$$F\text{-}measure_k = \frac{2 \times Recall_k \times Precision_k}{Recall_k + Precision_k} \tag{4.2}$$

where

$$Recall_k = \frac{TruePositives_k}{TruePositives_k + FalseNegatives_k} \tag{4.3}$$

and

$$Precision_k = \frac{TruePositives_k}{TruePositives_k + FalsePositives_k}. \tag{4.4}$$

Recall is a measure of how often a data point from a given class is recognized as being from that class, whereas *precision* is a measure of how often a data point recognized as being from a given class actually belongs to that class. The F-measure combines both recall and precision in a way that requires them both to be high and similar. Therefore, the new target is chosen based on the ANN's confidence in its outputs for a given class k as calculated by using the F-measure. The F-measure is attractive as a confidence measure because it takes into account both recall and precision, which are valid measures of an ANN's performance on a given class.

It was found in practice that when the ANN misclassifies a given example, setting the confidence $C_{i,j}$ to 0 despite the F-measure results in improved results. This suggests that the 0/1 targets are still best when the ANN is struggling to learn

a data point. This is equivalent to setting $\alpha = 0$ when the ANN misclassifies an example. In addition, better results were obtained by setting $\alpha = 0.5$ when the ANN correctly classifies an example. This is most likely because the F-measure is still only a heuristic, and therefore can not be trusted completely as a measure of the probability that the ANN's exact output is correct. This results the in new target values lying between 0.5 and 1.0 for the target class, and 0.0 and 0.5 for the non-target classes. If α were always set to 1, then the targets could vary anywhere between 0 and 1 for both situations. Note that as long as the correct class had the highest output, it does not matter how high that output is. For example, if the ANN output corresponding to the correct class outputs 0.3 and that is still the highest output, the ANN has classified the example correctly. Combining these settings results in simplifying the targets when the ANN classifies correctly in order to leave more adaptive capacity in the ANN for learning the more difficult data points. The function presented becomes more complex than the pure SOL function, but uses the weights of the ANN more efficiently than pure 0/1 relabeling. The mixing of pure SOL and 0/1 targets is what leads to SOL-CTR's success. It is able to combine the merits of both to more effectively train ANNs.

Using the F-measure with SOL yields a learning algorithm that relabels targets for training ANNs with backpropagation. Unfortunately, as is, SOL requires the ANN to be entirely retrained each time the data is relabeled. Figure 4.2 shows a more efficient approach, *SOL-CTR by Epoch* (SOL-CTR-E). It was designed to train the ANN only once by updating the target labels after every epoch instead of after completely training the ANN.

An even more efficient method is to relabel the targets after each data point. This method is called *SOL-CTR by Pattern* (SOL-CTR-P) and can be seen in 4.3.

Notice that the only difference between SOL-CTR-P and SOL-CTR-E is that SOL-CTR-P adapts the targets *within* the training epoch whereas SOL-CTR-E only

Procedure `SOL-CTR-E`(*training set, hold-out set,* α)

$S_{i,j} = T_{i,j} = 0/1$ targets for data point i and output j
`// Initialize true positives (TP) false positives (FP) and false negatives (FN)`
$TP = FP = FN = \{0\}$
while *hold-out set accuracy increases* **do**
  Train ANN for one epoch using T.
  foreach *data point i in the training set* **do**
    `// Store the ANN outputs for later relabeling`
    foreach *ANN output j* **do**
      $O_{i,j} =$ ANN's jth output given i
    `// Store the ANN classification`
    $class = argmax(O_i)$
    `// Get the true classification`
    $k = argmax(T_i)$
    if $class = k$ **then** $TP_k = TP_k + 1$
    else
      $FP_{class} = FP_{class} + 1$
      $FN_k = FN_k + 1$
  foreach *class of the problem* **do**
    $Recall_{class} = \frac{TP_{class}}{TP_{class}+FN_{class}}$
    $Precision_{class} = \frac{TP_{class}}{TP_{class}+FP_{class}}$
    $C_{i,j} = F\text{-}measure_{class} = \frac{2 \times Recall_{class} \times Precision_{class}}{Recall_{class}+Precision_{class}}$
  foreach *data point i in the training set* **do**
    foreach *target j of point i* **do**
      $T_{i,j} = \alpha C_{i,j} O_{i,j} + (1 - \alpha C_{i,j}) S_{i,j}$

Figure 4.2: SOL-CTR-E

Procedure `SOL-CTR-P`(*training set,hold-out set,*α)

$S_{i,j} = T_{i,j} = 0/1$ targets for data point i and output j
`// Initialize true positives (TP) false positives (FP) and false negatives (FN)`
$TP = FP = FN = \{0\}$
while *hold-out set accuracy increases* **do**
  foreach *data point i in the training set* **do**
    `// Store the ANN outputs for later relabeling`
    foreach *ANN output j* **do**
      $O_{i,j} =$ ANN's jth output given i
    `// Store the ANN classification`
    $class = argmax(O_i)$
    `// Get the true classification`
    $k = argmax(T_i)$
    if $class = k$ **then** $TP_k = TP_k + 1$
    else
      $FP_{class} = FP_{class} + 1$
      $FN_k = FN_k + 1$
    foreach *class of the problem* **do**
      $Recall_{class} = \frac{TP_{class}}{TP_{class}+FN_{class}}$
      $Precision_{class} = \frac{TP_{class}}{TP_{class}+FP_{class}}$
      $C_{i,j} = F\text{-}measure_{class} = \frac{2 \times Recall_{class} \times Precision_{class}}{Recall_{class}+Precision_{class}}$
    foreach *target j of point i* **do**
      $T_{i,j} = \alpha C_{i,j} O_{i,j} + (1 - \alpha C_{i,j}) S_{i,j}$
    Backpropagate error using T_i as the targets.

Figure 4.3: SOL-CTR-P

relabels the targets after the training epoch. In the last line of SOL-CTR-P, the newly generated targets are immediately used to train on the current pattern, whereas SOL-CTR-E will not apply the new targets until it trains for another epoch. Because the F-measure information is less accurate until the ANN's initial class accuracy trends emerge, it is better in practice to train an epoch before relabeling the data when using SOL-CTR-P. In the preliminary experiments we used to guide our research, SOL-CTR-E and SOL-CTR-P resulted in improved accuracy and average final weight magnitudes 87% lower than training with standard 0/1 targets.

4.4 Methods

In order to test the effectiveness of SOL-CTR, both SOL-CTR-E and SOL-CTR-P are compared to using standard 0/1 targets on 37 UCI Machine Learning Database problems (MLDB), two versions of a large, real-world OCR data set consisting of 500,000 OCR examples, and one large automated-speech recognition (ASR) data set consisting of over 800,000 phonemes taken from the TIDIGITS speech corpus. The difference between the first and second version of the OCR data set is that noise is added to the images in the second set to increase the difficulty of the problem. Each character in the OCR set consists of an $8x8$ grid of grayscale pixel values. To obtain the noisy set, a different amount of noise was chosen for each pixel by randomly generating a number from a standard normal distribution, cubing it, dividing it by 3, and then adding it to the pixel's original value. Pixel values were then clipped to $[0, 1]$. This is repeated for each pixel on each character creating the effect seen in figure 4.4. The OCR sets are broken into a 200,000 example training set, a 100,000 example

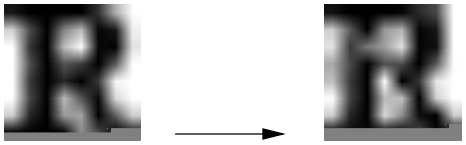

Figure 4.4: The character R before and after adding random noise

hold-out set, and a 200,000 example test set. The ASR data set was broken into a 500,000 phoneme training set, a 200,000 phoneme hold-out set, and a 200,000 test set. To verify the statistical significance of the results, a 10 pair permutation test is used [49] with each data set. For the MLDB problems, stratified 10-fold cross validation [40] is used, and therefore each pair is a different fold with a slightly different training set and a different test set in addition to different initial weights. For the ASR and OCR sets, a different initial weight setting is used for each pair. Different training set and test set partitions are not used with the larger sets since the size of the data sets is large enough to capture the distribution adequately [19]. Each pair consists of one ANN trained using a relabeling method, and one ANN trained using standard 0/1 targets—both ANNs use the same number of hidden nodes: 128, learning rate: 0.01, and have the exact same random initial weight settings.

4.5 Results and Analysis

Table 4.1 gives the results of the experiments. The first column gives the relabeling method being compared to training with the standard 0/1 targets and the following columns give the results of comparing the relabeling methods on 5 data sets. The *letter* data set from the MLDB has its results shown separately because its size (20,000 data points) is significantly larger than any other data set in the MLDB. The numbers given are the mean % differences in accuracy between the relabeling method and using standard 0/1 targets across the 10 pairs. The *'s give a measure of the statistical significance of the differences in accuracy. A single * means the resulting p-value from the experiment was below 0.05 whereas two *'s mean the resulting p-value was below 0.01. The results show that on the smaller data sets in the MLDB, SOL-CTR gives no significant difference in accuracy. However, on the letter data set, and on the other 3 large data sets, both relabeling methods consistently yield statistically significant improvements in accuracy over using standard 0/1 targets.

Table 4.1: % Differences in Mean Accuracy. *: p-value below 0.05, **: below 0.01.

Data / Method	SOL-CTR-E	SOL-CTR-P
MLDB	-0.0400	-0.0200
Letter	0.3000*	0.3100*
OCR	0.0520**	0.0560**
Noisy OCR	0.1872*	0.2002**
ASR	0.3140**	0.2630**

This suggests that SOL-CTR can successfully create an improved training function for backpropagation-trained ANNs, but perhaps only when enough data is available to effectively model the confidence of the ANNs. Although the average improvement on the large sets is modest at 0.05-0.3%, it is consistent and can be attributed to the relabeling. Further research improving the confidence measure can result in increased improvements because a more accurately modeled confidence measure means the targets can be more effectively chosen for training. If the targets are more effectively chosen, the resulting function can be even "easier" for backpropagation-trained ANNs to learn, and therefore resulting in even better accuracy.

The main contribution of the experiment is that it serves as a proof of concept for adapting the problem to the learner. Instead of only adapting the learner to the problem, machine learning can interactively use the feedback of a given learner to improve the problem's representation. In this manner, a given problem can be made to better fit the inductive bias of a given learner, and therefore broaden the set of problems over which a given learning algorithm performs well. Future research will develop problem adapting methods for additional machine learning algorithms besides backpropagation-trained ANNs.

4.6 Conclusions and Future Work

The results suggest that SOL-CTR can construct an easier function for backpropagation-training. Even more importantly, the results show the potential for using methods that adapt the problem to the learner instead of only adapting the learner to the problem. For future work we will investigate methods for learning improved targets that do not require a confidence metric. We will also develop problem adaption techniques for learning algorithms besides backpropagation-trained ANNs.

One way to learn targets without using a confidence measure is to iteratively refine them throughout the training process. Training would update both the weights and the targets simultaneously, attempting to minimize classification error as the objective.

Chapter 5

Additional Issues in Oracle Learning

Abstract

The following chapter examines the situations under which it is appropriate to use the versions of oracle learning discussed in previous chapters. It compares oracle learning in the presence of a larger oracle to using SOL-CTR. In addition, it compares these methods to standard training, weight decay, and basic self-oracle learning (SOL). These comparisons are made in situations where a smaller ANN is and is not desired, and in situations where unlabeled data is and is not available. Performance is measured across 35 datasets including a larger automated speech recognition (ASR) dataset, two optical character recognition (OCR) data sets, and 32 data sets from the UCI Machine Learning Database (MLDB) repository. The results show that when the goal is standard training and not reducing the size of an ANN, SOL-CTR is preferred when there are more available data and weight decay when there are less. When reducing the size of an ANN, SOL-CTR is preferred when there are no additional unlabeled data, but oracle learning in the presence of a larger oracle is the better method given unlabeled data.

5.1 Introduction

Chapter 2 introduces oracle learning for reducing the size of an ANN and chapter 4 gives the *self-oracle learning with confidence-based target relabeling* (SOL-CTR) method for use in general machine learning. However, neither chapter formally compares the two. At first it may appear that they are designed to solve different problems, however, as shown in 4.3.1, standard oracle learning can be used to retrain an ANN on self-relabeled data without confidence-based target relabeling, yielding the *self-oracle learning* (SOL) method. In addition, although SOL-CTR is not designed principally for reducing the size of ANNs, it does result in improved accuracy over standard training in the presence of enough data. Therefore, SOL-CTR can be used as an improved training method for training smaller ANNs directly. Since both methods can be used in similar situations, it is important to determine which is more appropriate for the situations under which they may be used.

For problems in which a large amount of unlabeled data is available, oracle learning may outperform SOL-CTR because it is able to label unlabeled data and take advantage of the additional information available. SOL-CTR can not make use of unlabeled data because it has no way of calculating the F-measure without knowing the true targets beforehand. However, without the presence of unlabeled data, the ability of SOL-CTR to preserve more information about the true targets may result in it yielding higher accuracy than standard SOL-CTR. The following chapter seeks to determine which method is best in each situation.

It has been suggested that the reason oracle learning has outperformed standard training even without the presence of unlabeled data is that it regularizes the weights of the ANN. The two most common methods of regularization in ANNs are early-stopping and weight decay [41]. The standard backpropagation training method employed in the experiments conducted in previous chapters did use early-stopping, but none employed weight decay. Therefore, it is important to determine if there are

situations under which it is preferable to use weight decay over the oracle learning methods. Chapter 4 showed that the accuracy of SOL-CTR did not improve significantly over standard training on the smaller data sets in the MLDB because there was not enough data to model the confidence of the ANN on the data. The performance of weight decay, however, does not depend on the size of the data set, and therefore using weight decay instead may result in better accuracy than the oracle learning approaches on smaller data sets. On larger sets, oracle learning methods may be preferable since the complexity of larger sets may require a form of regularization that is customized per weight, instead of being applied uniformly to all weights. In addition, weight decay is most effective when the decay rate is tuned specifically for each problem whereas oracle learning methods are not as sensitive to parameter tuning. There do exist methods for determining preferred weight decay rates [46], however they are second-order methods that would be intractable for larger ANNs on larger data sets.

The rest of this chapter serves to compare the oracle learning methods to each other and to weight decay on both large and small data sets, and on both standard training and training to reduce the size of ANNs. Section 5.2 further outlines the learning methods being compared, section 5.3 explains the experiments conducted to provide the comparisons, section 5.4 gives the results of the comparisons scenario by scenario, and section 5.5 gives conclusions and future work.

5.2 The Learning Methods

Five learning methods are compared to each other over several different situations. Here are the methods with brief descriptions. For full descriptions of the non-standard methods, see the references.

1. STD: This refers to the standard backpropagation training used in the previous chapters and is employed as the baseline. This method also uses early-stopping to prevent over-fit.

2. Decay and Decay001: None of the previous chapters have considered how weight decay compares to oracle learning. It has been suggested that the reason oracle learning has outperformed standard training even without the presence of unlabeled data is that it regularizes the weights of the ANN. The two most common methods of regularization in ANNs are early-stopping and weight decay [41]. The standard backpropagation training method employed here uses early-stopping, but not weight decay. It is expected that weight decay can outperform SOL-CTR on smaller sets because it can regularize without needing enough data to model the confidence of the training ANNs. On larger sets, oracle learning methods should be preferable since the complexity of those sets may require a form of regularization that is customized per weight, instead of being applied uniformly to all weights. In addition, weight decay is most effective when the decay rate is tuned specifically for each problem. This would have been too time-consuming for the scope of the following experiments. Therefore, the performance of weight decay shown may suffer because it has not been optimized for each of the 35 data sets used. On the other hand, the oracle learning methods used here are not as sensitive to parameter tuning and they customize the regularization effect per output instead of, as mentioned, uniformly. There do exist methods for determining preferred weight decay rates [46], however they are second-order methods that would be intractable for larger ANNs on larger data sets. Since changes in accuracy are sensitive to the weight decay rate chosen, two rates are used in the experiments. One at 0.00005, which will be labeled "Decay" and one at 0.0000001, which will be labeled "Decay001." The 0.00005 rate was chosen simply because it was the lowest rate applied in

[41] and therefore seemed a conservative choice. However, when early results showed 0.00005 may have been too large for certain data sets, the significantly smaller rate of 0.0000001 was chosen as an alternative.

3. OTN: OTN in this case stands for *oracle-trained network* and refers to the method presented in Chapter 2. For this chapter, a larger ANN serves as the oracle to re-label either the original data set or a set of data including both the original dataset and a set of effectively unlabeled data. Then, a smaller ANN trains on this larger set of data. This method is similar to the *self oracle-learning* (SOL) method described next, except in SOL, both the oracle and OTN are the same ANN.

4. SOL: SOL or *self-oracle learning* is the same as OTN except that both the oracle and OTN are the same ANN. This is also described in chapter 4. First, the ANN is trained on available labeled data. Then, the trained ANN is used to relabel this data and in some cases additional unlabeled data. Finally, the ANN's weights are reset back to the original starting configuration and the ANN is retrained on the self-relabeled data.

5. SOL-CTR: SOL-CTR or *self-oracle learning with confidence-based target relabeling* is exactly as described in Chapter 4. The main difference between this method and other forms of oracle learning is that it uses a confidence measure to balance between keeping the original labels and the oracle labels. Here, we use the SOL-CTR-P or *SOL-CTR by Pattern* method since its results have not been statistically different than the epoch-based version, and it is more efficient.

5.3 Description of Experiments

The purpose of this chapter is to determine how well the oracle learning methods compare to both each other and to weight decay in order to better understand when

each is appropriate. In order to determine this, the methods described in section 5.2 are compared across both large and small data sets, and in situations with and without unlabeled data. The experiments are also designed to show which method is preferable for reducing the size of an ANN in these situations.

There are two major scenarios the experiments consider. A scenario without any unlabeled data and a scenario where there are unlabeled data. The purpose of testing with no unlabeled data is to determine which method is most effective at improving accuracy in general. To do this, the full training sets are used although part of each training set is held-out for early stopping with all of the methods. In addition, separate test sets are used for the final results in each case.

The purpose for testing when there are unlabeled data is to see if how effective the oracle learning methods are at taking advantage of unlabeled data, and whether or not this can lead to a significant improvement in accuracy over the methods that do not use unlabeled data. To do this, only 25% of the original training set is allocated for learning. The OTN and SOL methods, however, are able to use their oracles to label the other 75% of the data. This experiment is similar to those used in chapter 2, except with the addition of the additional datasets and with the addition of comparisons to weight decay, SOL, and SOL-CTR.

Within each of the two major scenarios, the same two sub-cases are considered: a situation where there is no restriction on the size of the ANN and therefore large 128-hidden node ANNs are applied, and a situation that simulates a restriction on the size of the ANNs. This is to again determine which method is preferable in general training situations and which method is preferable for reducing the size of ANNs. By testing these sub-cases in both scenarios, it can be seen which method is best with and without unlabeled data, and for normal training versus training for size reduction.

Chapter 4 suggests that oracle learning methods may be more appropriate given larger or more complex data sets. Therefore, to also determine which method

is best in each of these scenarios as the size of the data sets change, each scenario and sub-case includes experiments on three large data sets and 32 smaller data sets. The larger data sets consist of an *automated speech recognition* (ASR) data set and two *optical character recognition* (OCR) data sets. One of the OCR datasets has had noise artificially added to it to increase the difficulty of the classification problem. This is described in more detail in chapters 3 and 4. The smaller datasets are taken from the UCI *machine learning database repository* (MLDB).

Previously, detailed experiments using the MLDB were not conducted because preliminary results suggested the data sets were either too small or too simple to show an improvement using oracle learning. Here this is verified on a set-by-set basis to empirically demonstrate this effect, and to verify if it is consistent. In chapter 2, the MLDB is not used because for many of the MLDB problems, there is no improvement in accuracy when going from a small to a large ANN. In this chapter, the larger OCR and ASR datasets use 32-hidden node ANNs for the smaller test ANNs and 128-hidden node ANNs as the larger ones. However, 32-hidden nodes may be large for most of the MLDB sets, and therefore in addition to using 32-hidden node ANNs, the following will also report results using 5-hidden node ANNs with the MLDB. This is in order to both demonstrate the fact that the size of the ANN does not seriously impact performance on those data sets, and to show how well the methods being compared here perform when the size of the ANN is significantly restricted relative to the MLDB data sets.

For the larger datasets, 10 ANNs with different random initial weights are trained as in chapter 4. The same weight settings are used across each method for these 10 ANNs. For the 32 UCI datasets, stratified 10-fold cross validation is used for each learning method. More in depth details of this experimental design can be found in chapter 4.

5.4 Results By Situation

Before continuing with the results section, consider table 5.1. This table compares standard (STD) training across the MLDB data sets using 128-, 32-, and 5-hidden node ANNs. Its purpose is to demonstrate that for many of the data sets, the size of the ANN does not affect its performance. This may be due to the difficulty of the problem in each set, or the amount of data available. All of the tables in this chapter use the reporting convention given in [12]. The method with the highest average test-set accuracy is shown in **boldface**. Methods whose accuracy is not statistically distinguishable from the best at $p = 0.05$ using the 10-pair stratified cross-validation permutation test given in chapter 6 are *'ed. Methods that are neither starred nor bold were significantly worse than the best methods at $p = 0.05$. Multiple boldface results indicate ties. For an addition summary, the bottom row in each table gives a *MLDB WTL* result, meaning a summary of the number of wins, ties, and losses for each method on the MLDB data sets only. A win is recorded if all other methods were significantly worse (no other method in bold or *'ed). A tie is recorded if the method either had the highest accuracy but was statistically indistinguishable from at least one other method (another method was in bold or *'ed), or the method was not the best but statistically indistinguishable from the best method (the method itself is *'ed). A loss is recorded if the method was significantly worse than the best method (neither bold nor *'ed).

The results in this table suggest improvements in size reduction can only be expected on the *anneal*, *audiology*, *credit-a*, *letter*, *primary-tumor*, *segment*, *soybean*, and *zoo* data sets. This is because those are the only data sets where there is a significant drop in accuracy as the size of the ANNs get smaller. Notice that for the *vehicle* data set, the smaller ANNs yield significantly *better* results than the larger ANN. This may be because early-stopping is not regularizing the network weights sufficiently enough to prevent over-fit. If this is the case, the regularizing effect of

either the oracle learning methods or weight decay may lead to improvements over standard training when using the larger 128-hidden node ANN.

The results of the experiments are summarized in table 5.2. This table gives the situation for each experiment along with how each method performed. The "% Labeled" column refers to how much of the data set was given to the methods as labeled data. Where only 25% was given, the other 75% was re-labeled by the OTN and SOL methods. The "ANN Size" column distinguishes the larger 128-hidden node ANN experiments from the smaller 5- and 32-hidden node experiments. The smaller hidden node experiments are grouped together because their results are similar. The "Data Size" column refers to whether the data sets used in each situation were the larger OCR and ASR sets, or the smaller MLDB data sets. There are also three levels of performance used in the table:

1. "Best" implies that there was evidence (p-value < 0.05 from a 10-fold cross-validated permutation test) that the method would yield consistently higher accuracy over each other method on each data set.
2. "Most Stable" is used where even though a method did not yield consistently higher accuracy than every other method, it was never statistically worse than the best method according to the 10-fold cross-validated permutation test used.
3. "Good" is used for a method that was neither "Best" nor "Most Stable" because although these methods were not statistically better than the other methods, there was little evidence to suggest the actual difference in accuracy was significant from a practical perspective.

Notice that on the smaller MLDB data sets, no method is consistently better than all other methods. Instead, there are 1 or 2 methods that are never worse than the other methods. Therefore, being the preferred method implies the method is "safe" or "stable" in the given situation compared to the other methods. Notice also that

Table 5.1: Results comparing 128-, 32-, and 5-hidden node ANNs on the MLDB data sets.

Dataset / # hidden	128	32	5
anneal	**0.7707**	0.7640*	0.7617
anneal.ORIG	**0.7718**	0.7640*	0.7617*
audiology	**0.7536**	0.7484*	0.2524
autos	**0.4625**	**0.4625**	0.4562*
balance-scale	0.8704*	0.8785*	**0.8801**
breast-cancer	**0.7174**	0.7069*	0.7173*
breast-w	**0.9707**	0.9693*	0.9678*
colic	0.8667*	0.8667*	**0.9167**
colic.ORIG	**0.6631**	**0.6631**	**0.6631**
credit-a	**0.6756**	0.6532	0.6260
credit-g	**0.7000**	**0.7000**	**0.7000**
diabetes	**0.7629**	0.7603*	0.7604*
glass	0.5879*	0.5842*	**0.5881**
heart-c	0.6038*	0.6006*	**0.6074**
heart-statlog	0.8296*	**0.8370**	0.8222*
hepatitis	**0.8208**	**0.8208**	0.8083*
ionosphere	**0.8721**	0.8663*	0.8692*
iris	0.9333*	0.9333*	**0.94**
kr-vs-kp	**0.9897**	0.9872*	0.9884*
letter	**0.9520**	0.8919	0.6096
lymph	0.7786*	**0.7919**	0.7719*
mushroom	0.9998*	**0.9999**	0.9998*
primary-tumor	**0.4660**	0.4574*	0.2655
segment	**0.9628**	0.961*	0.9498
sonar	0.7743*	0.7643*	**0.7833**
soybean	0.9283*	**0.9312**	0.7570
splice	0.9533*	**0.9545**	0.9526*
vehicle	0.4915	**0.5317**	0.5293*
vote	0.9517*	0.9518*	**0.9518**
vowel	**0.0909**	**0.0909**	**0.0909**
waveform-5000	0.8612*	**0.8638**	0.8572*
zoo	**0.9009**	0.8718*	0.8118
MLDB WTL	2-29-1	0-30-2	0-24-8

Table 5.2: Results Summary Table

% Labeled	ANN Size	Data Size	STD	Decay	OTN	SOL	SOL-CTR
100	Large	Large	Good	Good	Good	Good	Best
100	Large	Small	Good	Most Stable	Good	Good	Most Stable
100	Small	Large	Good	Good	Good	Good	Best
100	Small	Small	Good	Most Stable	Good	Good	Most Stable
25	Large	Large	Good	Best	Good	Good	Good
25	Large	Small	Good	Most Stable	Good	Good	Most Stable
25	Small	Large	Good	Good	Most Stable	Good	Good
25	Small	Small	Good	Good	Most Stable	Good	Good

although the practical significance of the comparisons on the larger, non-MLDB data sets is often small in terms of accuracy (sometimes as little as 0.05%), the results of the statistical significance tests suggest the chosen method will consistently outperform the other methods. This suggests that the preferred method is the "safest" choice in terms of yielding the best accuracy, even if that increase in accuracy is small.

Sections 5.4.1–5.4.2 give detailed results of each situation, considering them in the circumstances under which they would be used. The tables shown give the results in accuracy for each data set. For the MLDB sets, both the weights and the data are different across pairs. For the large data sets, only the weights are different between pairs. This is because the sets are large enough to infer the same results to similar data sets. However, the inference can still only be strictly applied to the data sets shown here. Using 35 data sets does, however, suggests that the chosen methods would show similar improvements on sets not used here.

5.4.1 No Unlabeled Data

For the first case, we consider the situation where there are no available unlabeled data points. To do this, the full training sets were used. Part of each training set was held-out for early stopping with all of the methods. Two sub-cases are considered as well. First, where there is no restriction on size and therefore a large, 128-hidden

node ANN is employed, and second where a smaller ANN is needed to simulate space or time constraints.

The purpose of testing this situation is to determine whether or not the oracle learning methods are effective at improving accuracy when there are no available unlabeled data, and if so which oracle learning method is best. The experiments show whether or not oracle learning can be used to improve accuracy in general.

No Size Restriction (128-hidden nodes)

For this case, five 128-hidden node ANNs, one for each method, were trained on each of the 35 data sets used in this experiments. The results are shown in table 5.3. Notice there is no OTN method listed here. This is because the OTN and SOL method are exactly the same in this case. A 128-hidden node ANN is trained on the data, used to relabel the data, and then reinitialized and trained on the relabeled data.

From these results, we see that the SOL-CTR method is consistently better than the other methods on the larger data sets, and at least as good as every other method on the smaller data sets. This is not too surprising considering it follows the observations in chapter 4, only adding comparisons to weight decay and SOL. Notice that the Decay001 method does at least as well as SOL-CTR on the MLDB problems, supporting the hypothesis that a correctly chosen decay rate should perform favorably when compared to SOL-CTR on smaller data sets because it can regularize without the need of enough data to model the ANN's confidence in the data.

Also, as observed in chapter 4, SOL-CTR shows more improvement on the larger datasets than the smaller ones, likely because there are enough data in those sets to learn the confidence of the ANN well enough. However, SOL-CTR is no worse than any other method on the 32 smaller MLDB sets, making it at least as stable as the other methods. This happens because SOL-CTR still preserves information about the true class, compared to SOL which uses the exact output of its oracle, even if the oracle

Table 5.3: Results using 128-hidden node ANNs on all of the data.

Dataset	STD	Decay	Decay001	SOL	SOL-CTR
ASR	0.8632	0.4733	0.8618	0.8635	**0.8658**
OCR	0.9708	0.4662	0.9706	0.9705	**0.9713**
NOISY-OCR	0.8688	0.3418	0.8685	0.8668	**0.8709**
anneal	0.7707*	**0.7740**	0.7729*	0.7718*	0.764*
anneal.ORIG	**0.7718**	0.7718*	0.7696*	0.7706*	0.7684*
audiology	**0.7536**	0.7132*	**0.7536**	0.7089	**0.7536**
autos	**0.4625**	0.4562*	0.425*	0.4375*	0.4313*
balance-scale	0.8704*	**0.8721**	0.8704*	0.8689*	0.8688*
breast-cancer	0.7174*	0.7104*	0.7174*	**0.7277**	0.7069*
breast-w	**0.9707**	0.9692*	**0.9707**	0.9692*	0.9692*
colic	**0.8667**	**0.8667**	**0.8667**	**0.8667**	**0.8667**
colic.ORIG	**0.6631**	**0.6631**	**0.6631**	**0.6631**	**0.6631**
credit-a	0.6756*	0.6681*	**0.6862**	0.6742*	0.6832*
credit-g	**0.7**	**0.7**	**0.7**	**0.7**	**0.7**
diabetes	0.7629*	**0.7707**	0.7591*	0.7616*	0.759*
glass	0.5879*	0.5933*	0.5833*	0.5699*	**0.6026**
heart-c	0.6038*	**0.6140**	0.6107*	0.5971*	0.6139*
heart-statlog	0.8296*	0.8259*	0.8296*	0.8296*	**0.8370**
hepatitis	**0.8208**	**0.8208**	0.7958*	0.8083*	0.8083*
ionosphere	0.8721*	0.8749*	0.8692*	0.872*	**0.8749**
iris	**0.9333**	0.9066*	**0.9333**	0.8933*	0.92*
kr-vs-kp	**0.9897**	0.9612	0.9878*	0.9850	0.9887*
letter	0.9520	0.6724	**0.9556**	0.9509	0.9551*
lymph	0.7786*	**0.7919**	0.7786*	0.7719*	0.7724*
mushroom	0.9998*	0.9974	**0.9999**	0.9996*	**0.9999**
primary-tumor	0.466*	0.475*	0.4693*	0.4483*	**0.4840**
segment	0.9628*	0.9104	**0.9641**	0.9567*	0.9636*
sonar	0.7743*	0.769*	0.7743*	0.774*	**0.8124**
soybean	0.9283*	0.9196*	0.9283*	0.9224*	**0.9312**
splice	0.9533*	**0.9571**	0.9545*	0.9545*	0.9558*
vehicle	0.4915*	0.4727*	**0.4975**	0.4763*	0.4927*
vote	0.9517*	0.9495*	0.9517*	0.9495*	**0.9563**
vowel	**0.0909**	**0.0909**	**0.0909**	**0.0909**	**0.0909**
waveform-5000	0.8612*	**0.8628**	0.8612*	0.8606*	0.8598*
zoo	**0.9009**	0.8618*	**0.9009**	0.8718*	0.8718*
MLDB WTL	0-31-1	0-28-4	0-32-0	0-29-3	0-32-0

is wrong. Note that although the plain SOL method appears to outperform standard training on the ASR data set, the difference is not statistically significant, and would not be expected to remain consistent. It appears the regularization benefits of oracle learning in the case of SOL do not outweigh the accuracy lost by approximating the oracle when it is wrong. SOL-CTR does not approximate the oracle when it is wrong, but uses the original targets, and this may be why it yields better results. These observations were what originally led to the development of the SOL-CTR method.

Note that even with the oracle learning and weight decay methods, the 128-hidden node ANNs are still significantly worse than the 32- and 5-hidden node ANNs on the *vehicle* data set. This suggests none of the regularization methods prevented the 128-hidden node ANN from over-fitting on this data set. Future research will investigate this anomaly to discover a more robust regularization method.

Size Restricted to 32- and 5-Hidden Nodes

Here the same experiments were performed except using 32-hidden node ANNs instead of 128-hidden node ones for the larger data sets, and both 32- and 5-hidden node ANNs for the smaller data sets. The purpose of this case is to determine which method is better for creating smaller ANNs if no unlabeled data is available. In this case an experiment using OTNs is included because instead of a 128-hidden node ANN approximating another 128-hidden node ANN, there are two cases. In one case, a 32-hidden node ANN approximates the 128-hidden node standard ANN from the previous section (OTN). In the other case (SOL) a 32-hidden node ANN approximates itself. Tables 5.4 and 5.5 show the results.

Again, the results are strongly in favor of the SOL-CTR method, which appears to be making best use of the ANN's parameters. On the larger data sets, SOL-CTR

Table 5.4: Results using 32-hidden node ANNs on all of the data.

Dataset	STD	Decay	Decay001	OTN	SOL	SOL-CTR
ASR	0.7939	0.3366	0.7933	0.7991	0.7886	**0.8037**
OCR	0.9151	0.2319	0.9147	0.9102	0.9003	**0.9205**
NOISY-OCR	0.7346	0.1881	0.7312	0.7287	0.7183	**0.7429**
anneal	0.764*	0.7617*	0.7651*	0.7629*	0.7629*	**0.7673**
anneal.ORIG	0.764*	0.7629*	0.7673*	0.7628*	0.7629*	**0.7706**
audiology	**0.7484**	0.6866*	0.7267*	0.6864*	0.682*	0.7354*
autos	**0.4625**	0.4437*	0.45*	0.4437*	0.4562*	0.4562*
balance-scale	**0.8785**	0.8753*	**0.8785**	0.8688*	**0.8785**	0.872*
breast-cancer	0.7069*	0.7069*	0.7069*	**0.7207**	0.7173*	0.6962*
breast-w	**0.9693**	0.9663*	**0.9693**	0.9663*	0.9649*	0.9649*
colic	**0.8667**	**0.8667**	**0.8667**	**0.8667**	**0.8667**	**0.8667**
colic.ORIG	**0.6631**	**0.6631**	**0.6631**	**0.6631**	**0.6631**	**0.6631**
credit-a	0.6532	0.6442	0.6411	0.6727*	**0.6923**	0.6621*
credit-g	**0.7**	**0.7**	**0.7**	**0.7**	**0.7**	**0.7**
diabetes	0.7603*	**0.7707**	0.759*	0.7616*	0.7642*	0.7629*
glass	0.5842*	0.5983*	0.5842*	0.5658*	0.5608*	**0.6024**
heart-c	0.6006*	0.6072*	**0.6306**	0.5971*	0.614*	0.6107*
heart-statlog	0.837*	0.837*	0.837*	0.8185*	**0.8407**	0.837*
hepatitis	**0.8208**	**0.8208**	0.7958*	0.8083*	**0.8208**	**0.8208**
ionosphere	0.8663*	0.872*	0.8663*	0.8663*	0.8635*	**0.8778**
iris	**0.9333**	0.9133*	**0.9333**	0.8867*	0.8867*	0.92*
kr-vs-kp	0.9872*	0.9593	**0.9875**	0.984*	0.9847*	0.9865*
letter	0.8919*	0.6570	0.8908*	0.8884*	0.8885*	**0.8921**
lymph	**0.7919**	0.7719*	**0.7919**	0.7647*	0.7586*	0.7452*
mushroom	**0.9999**	0.9974	**0.9999**	0.9998*	0.9998*	**0.9999**
primary-tumor	0.4574*	0.466*	0.4603*	0.4336*	0.4544*	**0.4693**
segment	0.961*	0.9138	**0.9637**	0.9598*	0.9554	0.9632*
sonar	0.7643*	0.7786*	0.7643*	0.764*	0.774*	**0.7978**
soybean	0.9312*	0.9195*	**0.94**	0.9195*	0.9136	**0.94**
splice	0.9545*	0.9567*	0.9545*	0.9539*	0.9555*	**0.9571**
vehicle	**0.5317**	0.4645	0.5188*	0.4740	0.5023*	0.5175*
vote	**0.9518**	0.9495*	**0.9518**	0.9471*	0.9495*	0.9495*
vowel	**0.0909**	**0.0909**	**0.0909**	**0.0909**	**0.0909**	**0.0909**
waveform-5000	0.8638*	**0.8666**	0.8628*	0.8608*	0.8604*	0.8618*
zoo	**0.8718**	0.8618*	**0.8718**	0.8418*	0.8118*	0.8618*
MLDB WTL	0-31-1	0-26-6	0-31-1	0-31-1	0-30-2	0-32-0

Table 5.5: Results using 5-hidden node ANNs on all of the data for the MLDB sets only.

Dataset	STD	Decay	Decay001	OTN	SOL	SOL-CTR
anneal	0.7617*	0.7606*	0.7617*	0.7617*	**0.7618**	0.7617*
anneal.ORIG	0.7617*	0.7606*	0.7617*	0.7617*	**0.7618**	0.7617*
audiology	0.2524*	0.2524*	0.2524*	0.2524*	**0.2611**	0.2524*
autos	**0.4562**	0.3438	0.45*	0.3250	0.3	0.4375*
balance-scale	0.8801*	**0.8849**	0.8801*	0.8785*	0.8833*	0.8817*
breast-cancer	0.7173*	0.7171*	0.7138*	**0.7174**	0.7104*	0.7033*
breast-w	0.9678*	0.9692*	0.9678*	**0.9707**	0.9692*	0.9693*
colic	**0.9167**	**0.9167**	**0.9167**	**0.9167**	**0.9167**	**0.9167**
colic.ORIG	**0.6631**	**0.6631**	**0.6631**	**0.6631**	**0.6631**	**0.6631**
credit-a	0.626*	0.6245	0.6245*	0.6516*	**0.6545**	0.6215*
credit-g	**0.7**	**0.7**	**0.7**	**0.7**	**0.7**	**0.7**
diabetes	0.7604*	0.7668*	0.7604*	0.7603	**0.7694**	0.7655*
glass	0.5881*	0.5838*	0.5881*	0.6024*	**0.6076**	0.6026*
heart-c	0.6074*	**0.6240**	0.6107*	0.6074*	0.6074*	0.604*
heart-statlog	0.8222*	**0.8296**	0.8222*	0.8111*	0.8222*	0.8185*
hepatitis	0.8083*	**0.8208**	**0.8208**	**0.8208**	**0.8208**	0.8083*
ionosphere	0.8692*	0.8663*	0.8692*	**0.8803**	0.8694*	0.8692*
iris	**0.94**	0.9333*	0.9333*	0.8533	0.8867*	0.8867*
kr-vs-kp	**0.9884**	0.9599	0.9881*	0.9869*	0.9859*	0.9881*
letter	0.6096*	0.2869	0.6089*	0.6134*	0.6082*	**0.6191**
lymph	0.7719*	0.7652*	0.7719*	0.7314*	0.7252*	**0.7786**
mushroom	0.9998*	0.9982	**0.9999**	0.9998*	0.9996*	**0.9999**
primary-tumor	0.2655*	0.2685*	0.2655*	0.2657*	0.2566*	**0.2715**
segment	**0.9498**	0.9147	0.9476*	0.9342	0.9433*	0.9494*
sonar	0.7833*	**0.7928**	0.7833*	0.7738*	0.7786*	0.7783*
soybean	0.757*	0.3133	0.7526*	0.6781	0.6514	**0.8270**
splice	0.9526*	**0.9574**	0.9526*	0.9523*	0.9501	0.9552*
vehicle	0.5293*	0.4610	**0.5352**	0.4845*	0.4998	0.5175*
vote	0.9518*	0.9517*	0.9518*	0.9449*	0.9471*	**0.9609**
vowel	**0.0909**	**0.0909**	**0.0909**	**0.0909**	**0.0909**	**0.0909**
waveform-5000	0.8572	0.8614*	0.8594*	0.8628*	0.8594*	**0.8666**
zoo	0.8118*	0.8127*	0.8118*	0.7918*	0.7227*	**0.8209**
MLDB WTL	0-31-1	0-24-8	0-32-0	0-27-5	0-28-4	0-32-0

is still the clear winner. On the smaller MLDB data sets, the SOL-CTR method is never significantly better than all of the methods, but also never worse than the best method. This again suggests it is a stable method. Also, the lack of distinction on the smaller data sets is not unexpected, as shown at the beginning of section 5.4. The decrease in accuracy when going from a larger to smaller ANN is not large on the MLDB problems. Of the data sets pointed out at the beginning of this section, the only data set that shows an improvement over standard training for creating smaller ANNs is the *credit-a* set, and this is only when using a 32-hidden node ANN. For every other case, the results are in generally even across each method.

It is also interesting to note that although the OTN method does not outperform SOL-CTR on the ASR data set—again apparently because of keeping the correct targets when the oracle is wrong—the OTN method *does* outperform standard training on the ASR data set. Although it is not shown here, the difference is statistically significant. In this case, approximating the more correct 128-hidden node ANN resulted in a more accurate ANN despite there not being any extra unlabeled data. This is therefore most likely due to the same regularizing effect that allows the SOL-CTR method to use the weights more efficiently.

As for weight decay, the large difference in accuracy between the two rates shows that it is indeed quite sensitive to its parameter setting. Part of the win in using a method like SOL-CTR instead of weight decay is that it is less sensitive to parameter tuning.

The improvements over standard training on the larger data sets using 32-hidden node ANNs are larger, which follows the results found in chapter 2. One possible reason for a larger improvement using smaller ANNs is that regularization becomes more important when there are less weights to adapt. Having less parameters means those that are available need to be used more efficiently, and regularization serves this purpose. The customized regularization provided by the oracle learning

methods apparently serves this purpose better than weight decay under these circumstances.

The results of this section answer the question as to which method between OTN and SOL-CTR is better for reducing the size of an ANN where no unlabeled data is available. It appears that SOL-CTR is the preferred method.

5.4.2 25% Labeled and 75% Unlabeled Data

In this section, the goal is to evaluate these methods again, but assuming less labeled data was originally available. To do this, only 25% of the original training set is allocated for learning. The OTN and SOL methods, however, are able to use their oracles to label the other 75% of the data. This experiment is similar to those used in chapter 2, except we are adding in the noisy OCR data set and the MLDB data set, in addition to comparisons with weight decay, SOL, and SOL-CTR.

No Size Restriction (128-hidden nodes)

As in the previous section, these first experiments show the results of using larger ANNs with no size reduction. Again, for the same reasons, we have no OTN method. However we do have two SOL methods. One where the self-oracle ANN is used to relabel only the 25% of the data that originally had labels, and one where it is used to also label the other 75% of the data that has no training labels. The results are shown in table 5.6.

In these experiments the results of the second weight decay rate and that of SOL-CTR trade places. The weight decay method is shown superior for the large sets, and both the weight decay and SOL-CTR methods are stable on the MLDB sets. Apparently, with only 25% of the data available, the SOL-CTR method can not model the confidence of the ANN well enough to compete with a well-chosen decay

Table 5.6: Results using 128-hidden node ANNs on 25% of the data.

Dataset	STD	Decay	Decay001	SOL 25%	SOL 100%	SOL-CTR
ASR	0.8334	0.4746	**0.8469**	0.8335	0.8338	0.8352
OCR	0.9605	0.4645	**0.9637**	0.9602	0.9604	0.9609
NOISY-OCR	0.8370	0.3472	**0.8446**	0.8362	0.8367	0.8392
anneal	0.7595*	0.7595*	0.7584*	0.764*	0.7573*	**0.7662**
anneal.ORIG	0.7617*	0.7606*	0.7606*	0.7595*	0.7606*	**0.7640**
audiology	0.6199*	0.598*	0.6154*	0.6067*	0.5628	**0.6201**
autos	0.4562*	**0.4688**	0.425*	0.4437*	0.4375*	0.4313*
balance-scale	0.8736*	**0.8768**	0.8736*	0.8752*	0.8736*	0.8752*
breast-cancer	0.7033*	0.7033*	0.7033*	**0.7171**	0.7028*	0.7064*
breast-w	0.9619*	0.9619*	0.9619*	0.9663*	0.9619*	**0.9678**
colic	**0.8667**	**0.8667**	**0.8667**	**0.8667**	**0.8667**	**0.8667**
colic.ORIG	**0.6631**	**0.6631**	**0.6631**	**0.6631**	**0.6631**	**0.6631**
credit-a	0.6772*	0.6833*	0.6787*	0.6742*	0.6803*	**0.6863**
credit-g	**0.7**	**0.7**	**0.7**	**0.7**	**0.7**	**0.7**
diabetes	0.7473*	0.746*	0.7473*	**0.7486**	0.746*	0.746*
glass	0.5045*	0.5141*	0.5141*	0.5232*	0.5139*	**0.5323**
heart-c	**0.6238**	0.6071*	0.6173*	**0.6238**	0.6104*	0.6206*
heart-statlog	0.8148*	0.8148*	0.8148*	**0.8222**	0.7963*	0.8037*
hepatitis	0.8208*	**0.8319**	0.8208*	0.8208*	0.8208*	0.8208*
ionosphere	0.8718*	0.8747*	0.8718*	**0.8832**	0.869*	0.8717*
iris	0.8533*	0.8533*	0.8533*	0.8533*	0.8333*	**0.86**
kr-vs-kp	**0.9631**	0.9521	0.9615*	0.9621*	0.9587*	0.9603*
letter	0.8994	0.6660	**0.9024**	0.8974	0.8959	0.8993*
lymph	**0.8052**	**0.8052**	**0.8052**	0.7586*	0.7652*	0.7852*
mushroom	0.9988*	0.9964*	0.9988*	**0.9989**	0.9988*	**0.9989**
primary-tumor	0.3984*	**0.3984**	0.3984*	0.3893*	0.3627*	0.3981*
segment	**0.9242**	0.9009*	0.9238*	0.9177*	0.9143	0.9173*
sonar	**0.7590**	0.7543*	**0.7590**	**0.7590**	0.7305*	0.7305*
soybean	0.8681*	0.8667*	0.8667*	0.8667*	0.8608*	**0.8682**
splice	0.9317*	0.9329*	0.9317*	**0.9367**	0.9313*	0.9317*
vehicle	**0.4489**	0.4453*	0.4443*	0.4418*	0.4323*	0.4312*
vote	0.9403*	0.9449*	0.9403*	0.9357*	0.9151*	**0.9450**
vowel	**0.0909**	**0.0909**	**0.0909**	**0.0909**	**0.0909**	**0.0909**
waveform-5000	0.847*	0.8488*	0.847*	0.8482*	0.847*	**0.8498**
zoo	**0.8209**	**0.8209**	**0.8209**	0.7718*	0.7027	0.8209*
MLDB WTL	0-31-1	0-30-2	0-32-0	0-28-4	0-31-1	0-32-0

rate. This follows the initial intuition that weight decay could outperform SOL-CTR given less data.

The weight decay method also consistently outperforms the SOL methods that relabel 100% of the data, despite having less data. It appears that keeping correct target information is better than having more data if the relabeling quality of that data can not be guaranteed. The results suggest that weight decay is the preferable method for standard training when less data is available

It is interesting to note that the SOL method using 100% of the data is consistently better than using only 25% of the data on the larger data sets. This is not surprising given the increase in available unlabeled data, and concurs with the conclusions made in chapter 2. Although not apparent in the table, the differences are statistically significant.

Size Restricted to 32- and 5-Hidden Nodes

The last experiment conducted repeats the last section except it uses 32- and 5-hidden node ANNs instead of 128-hidden node ANNs to examine the effect of decreasing the size of the ANN on the performance of the methods examined. The 5-hidden node ANNs are only trained on the MLDB because the difference between a 128- and 32-hidden node ANN for the larger sets is significant enough, whereas it is not for many of the MLDB sets. Also, these results show the additional OTN column because now there is a 128-hidden node oracle used to relabel data for a 32-hidden node ANN. The two SOL ANNs are also still present. Note that this is again similar to the experiments conducted for chapter 2, except adding new methods and additional data sets. The results are shown in tables 5.7 and 5.8

Here there is no clear winner across the 3 larger data sets, although the OTN method is the only method never worse than the best method, and is the best method

Table 5.7: Results using 32-hidden node ANNs on 25% of the data.

Dataset	STD	Decay	Decay001	OTN	SOL 25%	SOL 100%	SOL-CTR
ASR	0.7791	0.3377	0.7854	**0.7957**	**0.7957**	0.7779	0.7908
OCR	0.9099	0.2334	**0.9150**	0.9101*	0.8918	0.8941	0.9147*
NOISY-OCR	0.7284	0.1814	0.7297	0.7322*	0.7137	0.7150	**0.7369**
anneal	0.7595*	**0.7640**	0.7584*	0.7629*	0.7606*	0.7584*	0.7606*
anneal.ORIG	0.7584*	0.7606*	0.7573*	**0.7640**	0.7606*	0.7573*	0.7606*
audiology	0.5664*	0.5263	0.5621*	**0.5933**	0.4379	0.5225*	0.5625*
autos	0.4313*	**0.4750**	0.4562*	0.4375*	0.4437*	0.45*	0.4375*
balance-scale	0.872*	**0.8768**	0.872*	0.8736*	0.8592*	0.8704*	0.8656*
breast-cancer	0.7031*	0.6995*	0.7031*	0.7137*	0.7099*	**0.7171**	0.7102*
breast-w	0.959*	0.9561*	0.959*	0.9575*	0.9546*	**0.9634**	0.9561*
colic	**0.8667**	**0.8667**	**0.8667**	**0.8667**	0.7667*	0.8167*	**0.8667**
colic.ORIG	**0.6631**	**0.6631**	**0.6631**	**0.6631**	**0.6631**	**0.6631**	**0.6631**
credit-a	0.6756*	0.6816*	**0.6832**	0.6742*	0.6681*	0.6816*	0.6816*
credit-g	**0.7**	**0.7**	**0.7**	**0.7**	**0.7**	**0.7**	**0.7**
diabetes	0.7434*	0.7447*	0.7434*	**0.7486**	0.7434*	0.7408*	0.7447*
glass	**0.5372**	0.5234*	**0.5372**	0.5232*	0.4634*	0.5006*	0.5232*
heart-c	0.6106*	0.614*	0.6074*	**0.6274**	0.6102*	0.6171*	0.614*
heart-statlog	0.8407*	0.8407*	0.8407*	0.8296*	0.8222*	0.8333*	**0.8481**
hepatitis	0.8083*	0.8083*	0.8083*	**0.8208**	**0.8208**	0.8083*	0.8083*
ionosphere	0.8633*	0.8633*	0.8633*	**0.8803**	0.8633*	0.8747*	0.8605*
iris	0.8733*	0.8533*	0.8733*	0.88*	0.8*	0.8533*	**0.8867**
kr-vs-kp	**0.9634**	0.9534*	0.9634*	0.9612*	0.9596*	0.9631*	0.9621*
letter	0.8401	0.6511	0.8430	**0.8631**	0.8374	0.8387	0.8431
lymph	**0.7857**	**0.7857**	**0.7857**	0.7509*	0.7057*	0.759*	0.7724*
mushroom	0.9985*	0.9963	0.9985*	**0.9988**	0.9985*	0.9985*	0.9986*
primary-tumor	0.3455*	0.2891	0.3514*	**0.3717**	0.2389	0.3098*	0.3335*
segment	0.9316*	0.9034	**0.9320**	0.9225*	0.9195	0.9299*	0.9268*
sonar	0.7352*	0.74*	0.7352*	**0.7638**	0.7402*	0.7402*	0.7257*
soybean	**0.8653**	0.8563*	0.8638*	0.8608*	0.8506*	0.855*	0.8638*
splice	0.9307*	0.9292*	0.931*	0.9364*	0.9298*	**0.9370**	0.931*
vehicle	0.4455*	0.423*	0.4419*	0.4453*	0.3982*	**0.4478**	0.4477*
vote	0.9357*	**0.9379**	0.9357*	0.9334*	0.915*	0.9334*	0.9357*
vowel	**0.0909**	**0.0909**	**0.0909**	**0.0909**	**0.0909**	**0.0909**	**0.0909**
waveform-5000	0.8478*	0.8504*	0.8488*	0.8496*	0.848*	0.8468*	**0.8518**
zoo	**0.8027**	**0.8027**	**0.8027**	0.7527*	0.6136	0.7227*	0.7727*
MLDB WTL	0-31-1	0-27-5	0-31-1	1-31-0	0-27-5	0-31-1	0-31-1

Table 5.8: Results using 5-hidden node ANNs on 25% of the data for the MLDB sets only.

Dataset	STD	Decay	Decay001	OTN	SOL 25%	SOL 100%	SOL-CTR
anneal	**0.7629**	**0.7629**	**0.7629**	0.7617*	0.7618*	**0.7629**	**0.7629**
anneal.ORIG	0.7618*	0.7618*	0.7618*	**0.7629**	0.7618*	0.7618*	0.7618*
audiology	0.2524*	0.2524*	0.2524*	0.2524*	0.2524*	0.2439*	**0.2526**
autos	0.2938*	0.2938*	0.2938*	**0.3375**	0.3*	0.2938*	0.2938*
balance-scale	0.8784*	0.8784*	0.8784*	0.8703*	0.8752*	0.8784*	**0.8816**
breast-cancer	0.7066*	0.7134*	0.7066*	**0.7206**	0.703*	0.703*	0.7134*
breast-w	0.9648*	**0.9692**	0.9648*	0.9678*	0.9634*	0.9648*	0.9678*
colic	**0.9167**	**0.9167**	**0.9167**	**0.9167**	**0.9167**	**0.9167**	**0.9167**
colic.ORIG	**0.6631**	**0.6631**	**0.6631**	**0.6631**	**0.6631**	**0.6631**	**0.6631**
credit-a	0.6712*	0.6636*	0.6682*	0.6561*	0.6726*	**0.6757**	0.6653*
credit-g	**0.7**	**0.7**	**0.7**	**0.7**	**0.7**	**0.7**	**0.7**
diabetes	0.7395*	**0.7486**	0.7408*	0.7486*	0.7434*	0.7396*	0.7434*
glass	0.5558*	0.5699*	0.5558*	0.5275*	0.4822*	0.5327*	**0.5740**
heart-c	0.6039*	0.5938*	0.6006*	0.6273*	**0.6274**	0.6006*	0.6071*
heart-statlog	0.8037*	0.8111*	0.8037*	**0.8222**	0.7593*	0.8185*	0.8148*
hepatitis	**0.8208**	**0.8208**	**0.8208**	**0.8208**	**0.8208**	0.8097*	**0.8208**
ionosphere	0.8575*	0.8575*	0.8575*	**0.8832**	0.8775*	0.8633*	0.8604*
iris	**0.9067**	0.9*	**0.9067**	0.82*	0.5933	0.8867*	0.8933*
kr-vs-kp	0.9656*	0.9543	**0.9659**	0.9634*	0.9603*	0.9643*	0.9618*
letter	0.5866	0.2592	0.5841	0.6072*	0.6015*	0.5863	**0.6117**
lymph	0.7652*	0.7586*	0.7652*	**0.7786**	0.7047*	0.7252*	0.7519*
mushroom	**0.9991**	0.9964	**0.9991**	0.9986*	0.9986*	**0.9991**	0.9986*
primary-tumor	0.2419*	0.2389*	0.2419*	**0.2804**	0.2478*	0.236*	0.2419*
segment	**0.9221**	0.9047*	0.9216*	0.9139*	0.9061	0.9178*	0.9138*
sonar	0.7452*	0.7405*	0.7452*	**0.7688**	0.7212*	0.7307*	0.7595*
soybean	0.3201	0.2573	0.2969	**0.6792**	0.3163	0.3056	0.4232
splice	0.9238*	0.9288*	0.9238*	**0.9348**	0.9307*	0.9295*	0.9279*
vehicle	0.4205*	0.3874*	0.4206*	**0.4430**	0.3746	0.4205*	0.4242*
vote	0.9426*	0.9426*	0.9426*	0.9311*	0.9173*	0.9335*	**0.9427**
vowel	**0.0909**	**0.0909**	**0.0909**	**0.0909**	**0.0909**	**0.0909**	**0.0909**
waveform-5000	**0.8480**	0.848*	**0.8480**	0.8464*	0.843*	0.848*	0.847*
zoo	0.4736*	0.4736*	0.4736*	**0.6245**	0.4154	0.3882	0.4355
MLDB WTL	0-30-2	0-28-4	0-30-2	1-31-0	0-27-5	0-29-3	0-30-2

on the ASR set. On the MLDB data sets, the OTN method is never worse than any other method, and is significantly better than every other method on the letter recognition data set. The results suggest that OTN is the preferable method for reducing the size of an ANN when given unlabeled data, which agrees with the conclusions in chapter 2.

These results are also in contrast to the previous ones in this section where weight decay was the clear winner. The main difference here is that in the last section, the 128-hidden node SOL-trained ANNs did not have a larger, more accurate oracle to label the unlabeled data. Here, the 32- and 5-hidden node OTNs have a 128-hidden node ANN to label the unlabeled data. Combining superior labels and unlabeled data is apparently enough, in some cases, to improve on training methods that do not take advantage of unlabeled data.

The results also suggests OTN may be able to surpass SOL-CTR in cases where keeping too much target information makes it difficult to efficiently use a smaller available set of weights. The OTN method is only approximating the oracle, instead of still attempting to reach near $0-1$ targets like SOL-CTR does. This may provide better regularization in these smaller ANNs. This may also be why the SOL method that uses only 25% of the data is able to match the OTN method using all of the data—it is regularizing just as well and does not need the extra data. The SOL method that uses 100% of the data is worse, however, perhaps because it is using a larger set of less accurately relabeled data.

This interesting result suggests the reason oracle learning results in greater accuracy as the size of the ANNs decreases (as also seen in chapter 2) is because the smaller sizes have less adaptive power and benefit more from regularization. Regularizing allows the ANN to more efficiently use the weights. This is also shown by the fact that the other regularizing method, weight decay, performs well on the OCR data set. Although it appears the more customized regularization provided to the

OTN yields better results. SOL-CTR is also still a strong performer, however OTN is always at least as good, and at times better and therefore it appears OTN is the preferred method when there are available unlabeled data and the size of the ANN needs to be reduced. This again supports the conclusions in chapter 2.

It is also worth noting that tuning the weight decay rates exhaustively for each data set may have yielded better results than those reported here.

5.5 Conclusion

The purpose of this chapter was to determine how well the oracle learning methods compare to both each other and to weight decay in order to better understand when each is appropriate. Experiments were conducted to compare these methods across both large and small data sets, and in situations with and without unlabeled data. The experiments were also designed to show which method was preferable for reducing the size of an ANN in these situations.

For training without a restriction on the size of the ANN, SOL-CTR appears to be the preferred method given the full data sets, and weight decay when there is less data available. When the goal is to reduce the size of the ANN, SOL-CTR is the preferred method when there are no unlabeled data, and OTN is preferable when there are unlabeled data. It also appears that the regularizing effect of the oracle learning methods in general may be as beneficial in some cases as having unlabeled data. Finally, it was confirmed that choosing the correct rate when using weight decay is indeed critical to its performance, and suggested that further tuning may yield results that better match those of the oracle learning methods.

As well as demonstrating when the oracle learning methods are preferable, these results also give insights as to how and why oracle learning works. There is apparently a trade-off in the oracle learning methods between the customized regularization effect of completely relabeling the targets, and retaining target information.

Where there are enough data to model an ANN's "confidence" in the data, SOL-CTR works best because it strikes a balance between using pure target information when the ANN is wrong, and using a customized form of regularization when the ANN is right. As shown in section 5.4.1, when there are enough data the customized regularization effects of SOL-CTR give better results than the uniform regularization of weight decay. In addition, retaining target information results in SOL-CTR showing an improvement over pure oracle learning in the same situation. The results reverse, however, given less data and a smaller ANN. The OTN method is more appropriate in this case, as shown in section 5.4.2. This is because the OTN method provides more regularization than SOL-CTR, which is important given a smaller ANN with less learnable parameters (weights). It frees up adaptability needed for a problem potentially more complex than the smaller model can represent. In addition, OTN also benefits from using unlabeled data, although the results suggest this is not necessary in all cases. For example, on the ASR set, the SOL method that did not use additional data performed as well as the OTN method that did. This shows that, in this case, the regularization effect of oracle learning was more important than having additional data. In conclusion, the oracle learning methods provide tools for striking a balance between pure customized regularization and retaining target information, and this chapter shows when each is more appropriate.

Further research in the area will take two directions. First, it is apparent that keeping target information may help in some cases and therefore continued research will determine if keeping the appropriate labels when the oracle is wrong on the labeled portion of the data increases accuracy. The second area of future research will follow those ideas at the end of chapter 4, namely better measuring an ANN's confidence. One additional method that will be investigated will be to combine the online Bayesian training method used in chapter 9 with multi-layer ANN training so that ANN confidence can be measured and weight updates given in a principled

Bayesian framework. This will result in specific learning rates for each weight of the ANN based on the variance of the weight, and should yield theoretically improved results in addition to statistical insights about the parameters and input variables themselves. This may also lead to an auto-regularizing training method more robust to situations like the *vehicle* data set where the smaller ANNs outperformed the larger ANN despite regularization.

Part III

Paired Comparisons

In addition to the work done on oracle learning, this dissertation also includes research in paired comparisons. Chapter 6 gives a method for comparing two different classifiers to each other. It is included here because algorithm and classifier comparison is an important topic in general for machine learning and neural network research. Principled methods of deciding whether to prefer one approach over another are necessary for any type of research that seeks to motivate the use of a novel learning algorithm. Chapter 6 suggests an approach for doing this that improves one of the more common methods for algorithm comparison—the pairwise t-test, giving both theoretical and empirical motivations for preferring it.

The other three chapters in this part of the dissertation shows how a system for rating individual players develops from an artificial neural network approach into a recursively updating Bayesian ranking system. This is related to paired-comparisons because it gives a method for comparing the pairing of two groups based on aggregating the skills of the individuals in each group.

Chapter 7 shows the initial neural network approach to predicting the outcome of team-based online competitive games by estimating individual player ratings as weights within a single-layer network.

Chapter 8 gives a hierarchical Bayesian model for achieving this same end. The Bayesian approach is able to model not only the rating of individual players, but also the uncertainty of the system in the ratings. The BTANN model did not give variances for the player ratings which this model does. In addition, it is easier to analyze how aspects of the application affect the model when viewed from the point-of-view of a statistical model instead of as an ANN.

The MCMC method used to estimate the model parameters in chapter 8 is too inefficient to be used in a real-time situation. Chapter 9 gives a method that approximates the same estimation of the model parameters in real-time and shows how this can be used real-time to improve gameplay.

It is important to note that the audience for chapters 8 and 9 is different than that for the previous papers in this dissertation. The previous papers are motivated by their general appeal to improving machine learning. These last two papers however, are written for an application-oriented audience whose primary interest is both the significance of the application itself and how the proposed models can be used to positively impact the application. In this case, the papers motivate team-based online competitive games by their impact on the industry and show how the models given can be used to improve overall gameplay. That said, the models presented can also be applied in any situation where it is important to judge individual performance in group competitive settings, including team sports, employee productivity, corporation ranking, group publication significance, etc. In addition, the efficient method for fitting the Bayesian ratings model given in chapter 9 can be used as a general method for fitting similar Bayesian models and as a Bayesian training method for artificial neural networks.

In addition to having a different target audience, another important aspect of the last two papers is that since the second paper will be published close in time to the first paper, the models in the first paper are revisited in the second. Therefore, after reading the first paper, the careful reader can focus more on following from the second paper: the last two paragraphs of section 1; the last two paragraphs of section 1.1; all of sections 3.5, 4, and 5; the last paragraph of section 6.3 noting that the rest of section 6 is similar to section 6 in chapter 8 except that here it is pointed out that the suggestions in chapter 8 can now be applied in real-life; and all of section 7. Chapter 9 is still significant in its own right because without the derived update method it presents, the models from chapter 8 could not be used in the target real-world application.

Chapter 6

Using Permutations Instead of Student's t Distribution for p-values in Paired-Difference Algorithm Comparisons

Abstract

The paired-difference t-test is commonly used in the machine learning community to determine whether one learning algorithm is better than another on a given learning task. This paper suggests the use of the permutation test instead because it calculates the exact p-value instead of an estimate. The permutation test is also distribution free and the time complexity is trivial for the commonly used 10-fold cross-validation paired-difference test. Results of experiments on real-world problems suggest it is not uncommon to see the t-test estimate deviate up to 30-50% from the exact p-value.

6.1 Introduction

A popular test used to compare two learning algorithms is the 10-fold cross-validation paired-difference t-test [19, 64]. This test uses Student's t distribution to estimate a p-value representing the probability that the mean of the differences observed occurred randomly. If the resulting p-value is very low (usually below 0.10) it can be concluded that the observed difference is more than can be explained by random chance, and is therefore statistically significant. In the context of machine learning, this amounts to comparing how well algorithm A does compared to algorithm B on a particular learning problem characterized by a data set D. If the mean difference between algorithm A's performance and algorithm B's performance using data set D is statistically significant, there is support to prefer using algorithm A for that particular learning problem. For this reason, tests comparing two or more algorithms are important in validating the utility of machine learning algorithms and comparing them to each other in different problem domains.

The problem with the t-test is that it does not yield the exact p-value, but instead an approximation based on assumptions about the distribution of a paired-difference test. It is possible, however, to calculate the exact p-value in a trivial amount of time for the common 10-fold version of the paired-difference test using a permutation test. Since this test yields a more accurate p-value and is easily calculated, this paper suggests using the permutation test instead of approximating the p-value with Student's t distribution. The results of experiments on several real world problems show it is not uncommon for the t-test approximation to deviate as much as 30-50% from the true p-value.

6.2 Background

6.2.1 Statistical Issues

There are two assumptions associated with using Student's t distribution to calculate the p value:

1. The differences are normally distributed.
2. The differences are independent.

Since the t-test is robust to the first assumption, it often yields a reasonable approximation to the the true p-value which, in a paired-difference test, is:

$$p = \frac{n}{N} \tag{6.1}$$

where n is the number of ways the mean difference can be as extreme or more extreme (for a two-sided p-value) than the observed mean difference and N is the total number of possible reassignments of the paired-differences given the results. It is a measure of how often it is expected that a difference as extreme or more extreme than the observed difference occurs randomly. The p-value can also be calculated exactly in $O(2^n)$ time where n is the number of pairs. Although exponential, for the small amount of pairs usually used in the literature ($k = 10$), calculating the p-value exactly is not unreasonable. In fact, in statistics the calculation of the exact p-value is known as a permutation test and is often available in popular statistical packages.

Tables 6.1–6.3 show a simple example of how to calculate the p-value using a permutation test. Consider permutation 1 to be the original results of an experiment involving two algorithms being compared with, in this simple example, a 3-fold cross validation paired-difference test. Table 6.1 uses 3 of the possible 8 permutations to show how different permutations are created by swapping the results between algorithms. Notice the only difference between permutation 1 and permutation 2 in

table 6.1 is that the first row or fold results have been swapped. In the same table, the difference between permutations 1 and 3 is that in 3, both the first and second row's results have been swapped. Notice that swapping the results simply causes the sign on the difference to change, therefore, table 6.2 can show the 8 possible permutations giving only the change in the differences for each fold. For example, notice that the difference between permutations 1 and 5 in table 6.2 is that the sign on the third difference has changed, meaning the results for the third fold have been swapped. To obtain the p-value, the number of mean differences as extreme or more extreme than the observed data are counted up and used as n in table 6.3. In this case, assuming permutation 1 represents the original results, the number of permutations that yield a mean greater than or equal to 0.0012 or less than or equal to -0.0012 are counted (for a two-sided p-value). The result includes permutations 1 (the original), 4, 5, and 8 for a total of 4. Finally, dividing 4 by the total number of possible permutations, 2^3 or 8 yields the p-value, 0.5 as shown in table 6.3. Since the p-value is high in this case, the mean difference observed is not considered statistically significant. If the p-value had been below 0.10, the difference would have been considered significant.

6.2.2 Past Research

Since being able to compare machine learning algorithms is one of the key elements in the research area, there have been several evaluations of popular practices, and suggestions for appropriate procedures. In [64], Reich evaluates common practices for comparing machine learning methods and suggests appropriate practices, including the use of the 10-fold cross-validation paired-difference t-test. Salzberg criticizes mainstream philosophies and statistical methods in machine learning in [67], especially when using statistics to compare algorithms. He also criticizes the use of Student's t-test in re-sampled approaches because the assumption of independence between samples is violated. Dietterich [19] supports Salzberg's criticism of the re-

Table 6.1: Permutation Test Example part 1

Algorithm A	Algorithm B	Difference
Permutation 1		
0.9330	0.9309	0.0021
0.9336	0.9315	0.0021
0.9302	0.9308	-0.0006
Mean		0.0012
Permutation 2		
0.9309	0.9330	-0.0021
0.9336	0.9315	0.0021
0.9302	0.9308	-0.0006
Mean		0.0002
Permutation 4		
0.9309	0.9330	-0.0021
0.9315	0.9336	-0.0021
0.9302	0.9308	-0.0006
Mean		-0.0016

Table 6.2: Permutation Test Example part 2

Permutation	Difference	Permutation	Difference
1	0.0021	5	0.0021
	0.0021		0.0021
	-0.0006		0.0006
Mean	0.0012	Mean	0.0016
2	-0.0021	6	-0.0021
	0.0021		0.0021
	-0.0006		0.0006
Mean	-0.0002	Mean	0.0002
3	0.0021	7	0.0021
	-0.0021		-0.0021
	-0.0006		0.0006
Mean	-0.0002	Mean	0.0002
4	-0.0021	8	-0.0021
	-0.0021		-0.0021
	-0.0006		0.0006
Mean	-0.0016	Mean	-0.0012

Table 6.3: Permutation Test Example part 3

n	N	p-value ($\frac{n}{N}$)
4	8	0.5

sampled paired t-test and also warns against the use of the 10-fold cross-validation paired-difference t-test. He offers a new t-test seeking to retain the power or ability of the 10-fold test to detect existing differences while improving on its error or tendency to detect non-existent differences. Dietterich also acknowledges that like the 10-fold test, his suggested test still violates the independence assumption. Kohavi [40] suggests a stratified approach to cross-validation that lessens the effect of non-independence by forcing the folds to retain the same distribution as the original data and yields more accurate p-values. In [54], Michaels further supports the use of stratified approaches because they represent the common "multi-modality" of classification error. He then suggests a unique method using replicate statistics that does not assume independence and results in improved confidence intervals that can be calculated efficiently.

Moving away from the usual paired approaches, [58] introduces a randomized ANOVA approach for comparing algorithms. In [60], Provost et. al. argue against the traditionally used classification accuracy for comparison and promote ROC analysis in its place. [6] gives an approach to using the area underneath ROC curves for evaluating machine learning algorithms. [76] shows the usage of ROC curves with artificial neural networks. In, [47], Maloof explains that ROC approaches employ analysis of variance methods (ANOVA) to determine if the results of the ROC analysis are statistically significant. He then empirically compares the standard ROC ANOVA approach to the LABMRMC method [21] used commonly in the medical decision making community. He finds that the standard ROC ANOVA approach and the LABMRMC method can make different decisions as to the preferred learning

algorithm and recommends using LABMRMC-type techniques because they more accurately model the assumptions in a cross-validation experiment.

Although the 10-fold approach has received criticism in the above research, this paper focuses on using 10-fold cross-validation not because it is the best method, only because it is very common. Here, it is suggested that if 10-fold cross-validation is to be used anyways, the exact p-value might as well be calculated instead of an approximation.

6.3 Motivation

The most convincing reason to choose the permutation test instead of the t-test is because the p-value will be exact instead of approximated, thus yielding a more accurate prediction of how random a given result is.

There are two theoretical reasons for choosing one statistical test over another:

1. It is less likely to detect a difference when there is none.
2. It is less likely to miss a difference when there is one.

Since the permutation test yields the exact p-value, it will always, by definition, be less likely than the t-test to detect a difference where there is none or miss a difference where there is one.

Two practical considerations when choosing one statistical test over another are:

1. Is one statistic easier to calculate than the other in terms of time and space requirements?
2. Does the resulting conclusion change significantly enough in practice to choose one over the other?

Since the number differences usually used in a paired-difference test is 10, the amount of time and space is relatively small with only 1,024 permutations to evaluate. The

far from optimized implementation used for the experiments in section 6.5 runs in just over a tenth of a second. Therefore, the unanswered question is whether or not the p-values can differ enough in practice to promote the use of the permutation test. The rest of the paper seeks to answer this question giving both methods of calculating the p-value in section 6.4, then descriptions of experiments comparing both methods in section 6.5. Section 6.6 gives the experimental results and discusses them and section 6.7 contains the conclusion and suggestions for further research.

6.4 Methods

The 10-fold cross-validated paired-difference test is used here to explain how to calculate both the t-test's approximation to the p-value and the exact p-value. The technique here can be used for a k-fold experiment or even for a re-sampled paired t-test (criticized in [19, 67]), although the complexity grows exponentially in k or the number of times the data is re-sampled. The cross-validated paired-difference test is chosen in particular for its popularity among machine learning researchers (see [64]). The idea is if 10 differences are already calculated, the amount of work to determine the exact p-value is trivial.

6.4.1 Run the Experiments

The first part of a 10-fold cross-validation experiment is to obtain results for each fold. Each fold usually consists of two sub-experiments which can be paired because they vary only in the treatment in question. For example, two equivalent *artificial neural networks* (ANNs) trained on the same data in the same order, but with different random initial weight settings. Each ANN is trained and then tested and the difference in accuracy between the two is saved. The details behind selecting the training and testing sets for k-fold cross-validation can be found in [19] and [64].

6.4.2 Calculating p from t

After calculating the difference in accuracy between each of the 10 folds, the 10 resulting differences can be used to calculate either the t-test or exact p-value. First, to approximate the p-value, the t-statistic is calculated using the 10 differences:

$$t = \frac{\bar{x}}{SE_x} \tag{6.2}$$

where $\bar{x}$ is the mean of k differences where k is the number of differences—10 in this case—and SE_x or the standard error of x is

$$SE_x = \frac{\sigma_x}{\sqrt{k}} \tag{6.3}$$

where σ_x is the standard deviation of the k differences. Calculated this way, t is known to follow Student's t distribution with $k - 1$ degrees of freedom.

6.4.3 Calculating p exactly

In order to calculate the exact p-value, the mean differences of all possible assignments of the given results must be evaluated. Evaluating all possible assignments involves calculating the mean difference of all possible reassignments of the group membership on each fold as in the example from section 6.2.1. The process explained in section 6.2.1 can be restated as the pseudo-code in figure 6.1.

A given permutation can be characterized by whether or not each fold has its signed changed. This can be implemented as a binary number where each binary digit corresponds to whether or not one of the differences has its sign changed. For 10 folds, this yields all 10-digit binary numbers or 1,024 permutations. Trying each permutation is equivalent to counting to 1,024, calculating and comparing the mean differences for each permutation. The binary number can be mapped onto the dif-

1. given 10 differences
2. mean difference $\leftarrow$ mean(the ten differences)
3. $n \leftarrow 0$
4. $N \leftarrow 2^{10} = 1024$
5. repeat
6. get next permutation
7. if mean(abs(permutation differences)) $\geq$ abs(mean difference)
8. $n \leftarrow n + 1$
9. until all permutations tried
10. $p - value \leftarrow \frac{n}{N}$

Figure 6.1: One way to calculate the exact p value using a permutation test.

ferences using a mask, changing line 6 in figure 6.1 to the pseudo-code in figure 6.2. Therefore, calculating the 1,024 differences is done easily and in a trivial amount of time using the given approach.

1. for permutation = 1 to 1024
2. for pair = 1 to 10
3. if (permutation (bit AND) 2^{pair}) = 1
4. difference[pair] = -difference[pair]

Figure 6.2: One way to enumerate all permutations given 10 pairs of differences.

6.5 Experiment

To determine how well the p values generated by the permutation test compare with those resulting from a standard t-test, 10-fold cross-validation paired tests are conducted comparing the accuracy of two feed-forward multilayer perceptron (artificial neural networks or ANN) classifiers each with a single hidden layer. The data sets used are described in table 6.4. The table also includes information about the number of hidden nodes of the ANNs used in each experiment. Every ANN used a learning

rate of 0.05. The training set for each fold is split into a training and hold-out set. The ANNs are trained on only the training partition, and then tested after each iteration on the hold-out partition. Training ceases when there has not been an increase in accuracy on the hold-out set for a period of 100 iterations. The ANN weight configuration resulting in the highest hold-out set accuracy is then tested on the test partition of the fold and that number is used as the final result to be compared with the other ANN in the the same fold. The only variation within each pair is different random initial weight settings, and therefore the resultant p-values should be relatively large as there is no significant difference between the ANNs. The only other source of variation is the difference in each of the 10 folds which is part of the design of the 10-fold cross-validation paired-difference test (as explained in [19] and [64].

Table 6.4: Data sets used in the experiments.

Data set	size	features	classes	nodes
OCR	500,000	64	83	32
iris	150	4	3	5
pima	768	8	2	5
letter-recognition	20,000	256	26	32
wave21	300	21	3	5
bupa	345	6	2	5
glass	214	9	7	5

6.6 Results and Discussion

The resulting p-values are shown in table 6.5. The p-values are the same or close for the iris and OCR data sets, but up to 50% different for the others. Although the conclusions would still be the same for both statistics in these cases, if the question of significance was closer than in the experiment, the t-test would be more likely to incorrectly detect a difference where there was none or miss a true existing difference. For example, if the relative difference remains at around 30% between the t-test and

permutation test p-values, a true p-value of 0.1429—not usually held as significant—could yield a t-test p-value of 0.10, often held as significant in the literature. Since the results suggests the t-test p-value can deviate as much as 30-50% from the exact p-value, using the permutation test will result in finding less false significant values and missing less truly significant results.

Table 6.5: t-test and permutation test p values on several data sets.

Data set	t	exact	difference	relative difference
OCR	0.9847	0.9746	0.0101	1%
iris	1.0	1.0	—	—
pima	0.6783	1.0	-0.3217	32%
letter-recognition	0.0117	0.0176	-0.0059	34%
wave21	0.8534	0.7852	0.0682	9%
bupa	0.2091	0.5000	-0.2909	58%
glass	0.2443	0.5000	-0.2557	51%

6.7 Conclusions

The permutation test is suggested for use in place of the standard t-test for small (10 pairs) paired-difference cross-validation tests because it is more accurate and the t-test approximation can deviate significantly from the true p-value. A method of computing the permutation test p-value is given. The complexity of the permutation test is reasonable for small numbers of pairs and the resulting p-value is more precise and therefore less likely to detect significance where there is none, and more likely to detect significance if present.

Future work will investigate the theoretically expected difference between permutation and t-test calculated p-values. Also, a more thorough test comparing the error and power of the permutation test relative to the t-test is appropriate. 10-fold cross-validation is criticized as having a high error in [19]. It is interesting to investigate how much of this error can be decreased through exact vs. approximate

calculation of the resulting p-values, despite the fact that the error is more likely to come from the violated independence assumptions as discussed in [19, 67, 54].

Chapter 7

A Bradley-Terry Artificial Neural Network Model for Individual Ratings in Group Competitions

Abstract

A common statistical model for paired comparisons is the Bradley-Terry model. This research re-parameterizes the Bradley-Terry model as a single-layer *artificial neural network* (ANN) and shows how it can be fitted using the delta rule. The ANN model is appealing because it makes using and extending the Bradley-Terry model accessible to a broader community. It also leads to natural incremental and iterative updating methods. Several extensions are presented that allow the ANN model to learn to predict the outcome of complex, uneven two-team group competitions by rating individuals—no other published model currently does this. An incremental-learning Bradley-Terry ANN yields a probability estimate within less than 5% of the actual value training over 3,379 multi-player online matches of a popular team- and objective-based first-person shooter.

7.1 Introduction

The Bradley-Terry model is well known for its use in statistics for paired comparisons [7, 17, 16, 37]. It has also been applied in machine learning to obtain multi-class probability estimates [32, 77, 36]. The original model states that:

$$Pr(A \text{ defeats } B) = \frac{\lambda_A}{\lambda_A + \lambda_B}. \tag{7.1}$$

where λ_A and λ_B are both positive and subjects A and B compete in a paired-comparison. λ_A and λ_A represent the strengths of subjects A and B.

Bradley-Terry maximum likelihood models are often fit using methods like Markov Chain Monte Carlo (MCMC) integration or fixed-point algorithms that seek the mode of the negative log-likelihood function [37]. These methods however, "are inadequate for large populations of competitors because the computation becomes intractable." [28]. The following paper presents a method for determining the ratings of over 4,000 players over 3,379 competitions.

Arpad Elo, a chess master and physics professor, suggested an efficient approach to update the ratings of thousands of players competing in thousands of chess tournaments. [22] Now commonly referred to as the ELO Rating System, it has been used in the past by both the United States Chess Federation (USCF) and the World Chess Federation (FIDE). Most common large-scale rating systems in use today have roots in the ELO system. His method re-parameterizes the Bradley-Terry model by setting $\lambda_A = 10^{\theta_A}$ yielding:

$$Pr(A \text{ defeats } B) = \frac{1}{1 + 10^{-\frac{\theta_A - \theta_B}{400}}}. \tag{7.2}$$

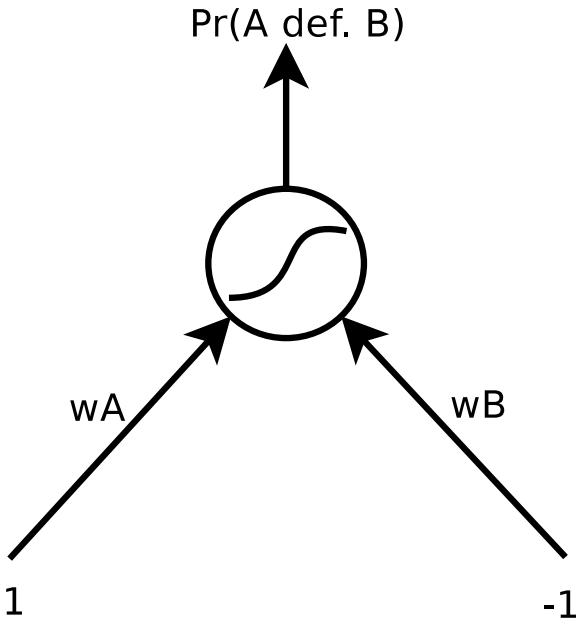

Figure 7.1: The Bradley-Terry Model as a Single-Layer ANN

The scale specific parameters are historical only and can be changed to the following which uses the natural base instead:

$$Pr(A \text{ defeats } B) = \frac{1}{1 + e^{-(\theta_A - \theta_B)}}. \tag{7.3}$$

This is also the equation for the standard sigmoid used as a transfer function in *artificial neural network* (ANN) nodes, and therefore substituting w for θ in (7.3) yields the single-layer ANN in figure 7.1. This single-layer sigmoid node ANN can be viewed as a Bradley-Terry model where the input corresponding to subject A is always 1, and B, always -1. The weights w_A and w_B correspond to the Bradley-Terry strengths θ_A and θ_B—often referred to as ratings. The Bradley-Terry ANN model can be fit by using the standard delta rule training method for single-layer ANNs.

This ANN interpretation is appealing because many types of common extensions can be added simply as additional inputs to the ANN. It also makes the Bradley-Terry model accessible to anyone familiar with ANNs and provides natural

incremental as well as iterative training methods (see section 7.3). Bradley-Terry models are useful because they can be applied to any situation where the probability of one subject being "better" than another is needed. This can include not only sports, but obtaining multi-class probability estimates from binary classifiers in machine learning [32, 77, 36], market predictions for economics, biostatistics [7], bibliometrics or deciding which publications are more significant [70], genetics [68], taste-testing, military encounters, and any other prediction that requires the probability of one subject being more desirable than another. In addition to making the usage of the Bradley-Terry model more accessible to the machine learning community, viewing the Bradley-Terry model as an ANN also makes research into extending and improving the model more accessible. This research demonstrates this by adding and proposing several extensions based on this ANN view of the Bradley-Terry model.

The first extension presented is actually a generalization of the original Bradley-Terry model which provides for the case where the subjects being compared are groups and the goal is to obtain the ratings of the individuals in the group. The original model does not do this. The following presents a model that can give individual ratings from groups and shows further extensions of the model to handle "home field advantage", weighting individuals by contribution, dealing with variable-sized groups, and other issues. The probability estimates given by the final model are accurate to within less than 5% of the actual observed values over 3,379 multi-player online matches of a popular team- and objective-based first-person shooter. This is significant because of the difficulty in the predicting the outcome of these highly dynamic matches. They can be compared to a soccer game where either team can have as many players as they like so long as the combined number of players on the field is less than some maximum. To complicate matters further, the players are free to switch between teams at will. Furthermore, the soccer field does not have to be symmetrical: the field can be on an incline such that the ball naturally rolls towards

one team's goal, and it can also include obstacles on one end that are not on the other. The pool of possible players also does not have a limit, ranging as high as 4,000, and the players are free to leave and join a match at any time. There are no other published models designed to handle this complex of a competition, and therefore nothing with which to compare the current results. The fact that the proposed model provides probability estimates within less than 5% of a match's actual outcome and that these estimates can be created in a short amount of time is in and of itself a significant result.

One other extension that will be presented is for dealing with the effects of the passage of time within a competition. There exist competitions and paired-comparisons in which the probability of a given competitor defeating the other is related to the amount of time that has passed in the competition. For instance, in chess, there may be players that win often when they are able to defeat their opponents early in the match, but who do not win as often when the match lasts longer than average. In the multi-player online game used for the results in this paper, one team is usually trying to prevent the other from achieving a known objective for a certain amount of time. When the objective is not accomplished in the usual amount of time, the probability of the achieving team winning is likely to be less than expected. Games and sports are not the only applications where the passage is significant. Modeling time in paired-comparisons can also be applied in education to determine how time affects different teaching and testing methods, it can be applied in medicine to account for the passage of time given different treatments, it can be used in military applications for balancing exercises or determining how long a given encounter should last, and it can be used in any other paired-comparison or competition where time may play an important role. One of the extensions in the following paper uses an artificial neural network approach to learn how important the passage of time is for

a given paired-comparison. Including this extension results in further a 16% decrease in error over the 3,379 matches.

This article will proceed as follows: section 2.2 gives more related background on prior uses and methods for the fitting of Bradley-Terry models, section 7.3 explains the proposed model, extensions, and how to fit it, section 7.4 explains how the model is evaluated, section 6.6 gives results and analysis of the model's evaluations, a further analysis of the meaning of the time weights is given in section 7.7, a discussion of possible applications of the model are given in section 7.6, and section 6.7 gives conclusions and future research directions.

7.2 Background

The Bradley-Terry model was originally introduced by Bradley and Terry [7]. It has been widely applied in situations where the strength of a particular choice or individual needs to be evaluated with respect to another. This includes sports and other competitive applications. There are several reviews on extensions and methods of fitting Bradley-Terry models [7, 17, 16, 37]. This paper differs from previous research in at least two ways. First, it proposes a method to determine individual ratings and rankings when the subjects being compared are groups. Recently, Huang et. al proposed similar extensions by modeling group ratings as the sum of the ratings of the individuals in each group [36, 35]. One of their papers models the increase in a larger group's rating linearly [36] and the other uses a similar model to that used here to model that increase exponentially [35]. We model group ratings as an exponentiated average of the individual ratings in a group, along with an exponentiated "field" weight that takes into account the number of individuals in each group. Like Huang et al's second model, ours assumes having more individuals in one group than another results in an exponential increase in the larger group's probability of winning. This is appealing in competitions where each individuals can act simultaneously.

The more common methods for fitting a Bradley-Terry model require that all of the comparisons to be considered involving all of the competitors have already taken place so the model can be fit simultaneously. This includes two different iterative methods proposed Huang et. al. [36, 35]. While this can be more theoretically accurate, it becomes computationally intensive in situations involving thousands of comparisons and competitors. It would require storing and retaining information about every match that ever occurs and refitting the model after each match. This can be unreasonable if the system is meant to continually processes a potentially unlimited amount of matches. In addition, as the number of matches grows, it may take longer to fit the model than the length of a match. An average match in the game used here usually lasts at least 15 minutes, but can often last as little as 5. Once the number of matches becomes large enough so that iterative methods take longer than this to fit the model, the rating estimates will fall farther and farther behind. This will make them unusable for the real-time applications discussed in section 7.6.

Elo [22], and also Glickman [28], both proposed methods for approximating an iterative fit by by adapting the Bradley-Terry model after every comparison without requiring storing match information. However, neither of their models has been extended to extract individual ratings from groups. The model presented here does allow individual ratings to be determined in addition to providing an efficient update after each comparison. Although adapting using only local information may result in theoretically less accurate estimates of the ratings than iterative methods, they do have the advantage of assuming a moving average of the ratings instead of a static one. This allows them to track time-varying changes in individual ratings. Over the course of thousands of matches, the true average of an individual rating may increase or decrease based on increased experience, or decrease in interest. Iterative methods such as those used by Huang et. al. [36, 35] assume individuals have the same rating over all matches. While this assumption may hold over a reasonable set or hundreds

of matches [35], it is less appropriate over a larger set or thousands of matches like that used here.

7.3 The ANN Model

The section proceeds as follows: the first section, 7.3.1, presents the basic framework for viewing the Bradley-Terry model as a delta-rule trained single layer ANN, section, 7.3.2 will show how to extend the model to learn individual ratings within a group, section 7.3.3 extends the model to allow different weights for each individual based on their contribution, section 7.3.4 shows how the model can be extended to learn "home field" advantages, section 7.3.6 presents a heuristic to deal with uncertainty in an individual's rating, and section 7.3.7 gives another heuristic for preventing rating inflation.

7.3.1 The Basic Model

The basis for the Bradley-Terry ANN Model begins with the ANN shown in figure 7.1. Assuming, without loss of generality, that the ANN model is designed to always predict the probability that subject A will win, the input for w_A will always be 1 and the input for w_B will always be -1. This assumption is a new approach for using ANNs because it suggests the use of interchangeable weights. It is an important part of viewing the Bradley-Terry model as an ANN because it allows a given subject to "carry its own weight" as its rating. That rating is "plugged in" as the winning or losing weight when that subject competes, and ignored otherwise. For example, given two subjects, 1 and 2, if subject 1 defeats subject 2, then w_A is set to the rating of subject 1, θ_1, and w_B is set to the rating of subject 2, θ_2. However, if subject 2 defeats subject 1, the opposite occurs: w_A is set to θ_2 because subject 2 is the winner. If a subject is not currently competing, then its weight or rating is ignored altogether—equivalent to setting the input for that weight to 0. Instead of the classical notion

that weights are always associated with the same inputs, the weights are interchanged based on the outcome of the comparison. Another way of viewing this approach is that the weight for every subject is always present, but the inputs are different. If a subject wins, the input for its weight becomes 1. If that subject loses, its input is -1. If the subject is not competing, its input is 0. In addition, only two weights can be non-zero for a given comparison—in other words, only two subjects are compared at a time. There do exist extensions to the Bradley-Terry model allowing for more than one comparison at a time [37, 30], but that is beyond the scope of this work.

The equation for the model can be written as:

$$Output = Pr(A \text{ defeats } B) = \frac{1}{1 + e^{-(w_A - w_B)}}. \tag{7.4}$$

which is the same as (7.3), except w_A and w_B are substituted for θ_A and θ_B. This can be rewritten as:

$$Output = Pr(A \text{ defeats } B) = \frac{e^{w_A}}{e^{w_A} + e^{w_B}}, \tag{7.5}$$

which is the same as (7.1) with $\lambda_A = e^{w_A}$. This is a conditional exponential model, like that of [35].

Since we assume that w_A is always the rating of the winning subject, and w_B the rating of the losing subject, the likelihood of w_A and w_B over a set of m matches can be written as:

$$L(w_A, w_B) = \prod_{i=1}^{m} \frac{e^{w_A}}{e^{w_A} + e^{w_B}}. \tag{7.6}$$

where w_{A_i} (w_{B_i}) is the rating of the winning (losing) subject in match i. The negative log-likelihood is therefore:

$$L(w_A, w_B) = -\sum_{i=1}^{m} log\left(\frac{e^{w_A}}{e^{w_A} + e^{w_B}}\right) \tag{7.7}$$

where the goal is finding the w_A and w_B for each i that minimize $L(w_A, w_B)$. As [35] stated, "it is well known that the log-likelihood of a conditional exponential model is concave. Thus [its negative log-likelihood] is convex, so one can easily find a global minimum..." Although [35] uses an iterative method to find the minimum of (7.7), they also mention that "Standard optimization methods (e.g., gradient...) can be used", which is what will be used here. This is done so that a sequential update can be derived that can minimize 7.7 over a potentially infinite set of matches.

The negative log-likelihood over the predictions of a set of Bernouli observations, like those given here, is also known as the cross-entropy of those predictions. A variation of the well-known delta rule for ANN training can be used to minimize the cross-entropy of the predictions of an ANN with respect to its weights. Further details and derivations can be found in [34, 57, 31]. For the ANN model used here, the delta rule update after a single match is:

$$\Delta w_i = \eta \delta x_i \tag{7.8}$$

where η is a learning rate, δ is the error measured with respect to the output, and x_i is the input for w_i. The error, δ is measured here as:

$$\delta = \text{Target Output} \ - \ \text{Actual Output}. \tag{7.9}$$

For the ANN Bradley-Terry model, the target output is always 1 given the assumption that A will defeat B. The actual output is the output of the single node ANN. Therefore, the delta rule error can be rewritten as:

$$\delta = 1 - Pr(A \text{ defeats } B) = 1 - Output, \tag{7.10}$$

and the weight updates can be written as:

$$\Delta w_A = \eta(1 - Output) \tag{7.11}$$

$$\Delta w_B = -\eta(1 - Output) \tag{7.12}$$

Here, x_i is implicit as 1 for w_A and -1 for w_B, again since A is assumed to be the winning subject. This formulation of the delta rule can be applied incrementally, updating the model after each competition, or iteratively over a training-set of paired-comparisons. The first approach may be less accurate, but is also less likely to overfit since it will never update based on the same comparison more than once. The iterative approach should only be used if the training-set can be assumed to contain a sampling of all possible competitions that matches all expected future competitions.

7.3.2 Individual Ratings from Groups

In order to extend the ANN model given in 7.3.1 to learn individual ratings, the weights for each group are obtained by averaging the ratings of the individuals in each group. Huang et. al [36, 35] modeled the strength of a given group by using the sum of the strengths of the individuals in the group. Their first model [36] assumes that when there are an uneven number of individuals in each group, the increase in rating from having a larger group is linear. Their second model [35] assumes the effect of a difference in the number of individuals is exponential. The former was appropriate for the application in [36] because they did not use their model for applications where there were an uneven number of individuals in each group, and therefore how their model handled uneven team numbers was not relevant. For the application in this paper, however, it is common to have competing groups of differing sizes. Instead of modeling the effect of an imbalance in numbers exponentially directly as Huang et. al. do [35], the model we present uses an average instead so that the effect of

a difference in the number of individuals can be handled separately. For example, the home field advantage formulation in section 7.3.4 uses the relative number of individuals per group as an explicit input into the ANN, yielding an exponential increase in a group's likelihood of winning if it has more individuals than the other group. Averaging instead of summing the ratings is also analogous to normalizing the inputs—a common practice in efficiently training ANNs. Neither method changes the general form of the likelihood. The only difference is that each subject is now a group instead of an individual. Instead of inserting an individual's known rating θ_i in for the winning or losing weight, the average of a group's individual ratings is inserted for the winning or losing weight. Instead of each individual "carrying its own weight" as a rating, each individual carries its own rating, and that rating is combined with the other individuals within the group to create that weight. Therefore, the weights for the ANN model in 7.1 are obtained as follows:

$$w_A = \frac{\sum_{i \in A} \theta_i}{N_A} \tag{7.13}$$

$$w_B = \frac{\sum_{i \in B} \theta_i}{N_B} \tag{7.14}$$

$$\tag{7.15}$$

Where $i \in A$ means individual i belongs to group A and N_A is the number of individuals in group A. With respect to the original Bradley-Terry model, the strength for a group A becomes:

$$\lambda_A = exp(\frac{\sum_{i \in A} \theta_i}{N_A}). \tag{7.16}$$

Combining individual ratings in this manner means that the difference in the performance between two different individuals within a single competition can not be distinguished. However, after two individuals have compared within several different groups, their ratings can diverge.

The weight update for each individual θ_i is equal to the update for Δw_{group} given in (7.11):

$$\forall i \in A, \Delta\theta_i = \eta(1 - Output) \tag{7.17}$$

$$\forall i \in B, \Delta\theta_i = -\eta(1 - Output) \tag{7.18}$$

where A is the winning group and B is the losing group. In this model, instead of updating w_A and w_B directly, each individual receives the same update, therefore indirectly changing the average group ratings w_i by the same amount that the update in (7.11) does for Δw_A and Δw_B.

Notice that this model still assumes that groups are fixed within a comparison. Section 7.3.3 discusses a way to implicitly model individuals changing their groups within a single comparison.

7.3.3 Weighting Player Contribution

Depending on the type of competition, prior knowledge may suggest certain individuals within a group are more or less important despite their ratings. As an example, consider a public online multi-player competition where players are free to join and leave either team at any time. All else being equal, players that remain in the competition longer are expected to contribute more to the outcome than those who leave early, join late, or divide their time between teams. Therefore, it would be appropriate to weight each player's contribution to w_A or w_B by the length of time they spent on team A and team B. In addition, the update to each player's rating should also be weighted by the amount of time the player spends on both teams. Therefore, w_A is rewritten as an average weighted by time spent in either group:

$$w_A = \sum_{i \in A} \theta_i \frac{t_A^i}{\sum_{j \in A} t_A^j}. \tag{7.19}$$

Here, t_A^i means the amount of time individual i spent in group A. The rating θ_i is therefore weighted by the amount of time individual i spends in group A relative to the total time spent by all individuals in group A. Weighting can be used for more than just the amount of time an individual participates in a competition. For example, in sports, it may have been previously determined that a given position is more important than another. This gives an average that is weighted by each individual's contribution compared to the rest of the individuals' contributions in the group.

7.3.4 Home Field Advantage

The common way of modeling home field advantage in a Bradley-Terry model is to add a parameter learned for each "field" into the rating of the appropriate group [1]. In the ANN model of figure 7.1, this would be analogous to adding a new connection with a weight of θ_f. If the advantage is in favor of the winning group, the input for θ_f is 1, if it is for the losing group, that input is -1. This would be one way to model home field advantage in the current ANN model. However, since the model uses group averages to form the weights, the home field advantage model can be expanded to include situations where there is an uneven amount of individuals in each group ($N_A \neq N_B$). Instead of having a single parameter per field, θ_{fg} is the advantage for group g on field f. The input to the ANN model then becomes 1 for the winning group (A) and -1 for the losing group (B). The parameters can be stored one per group per field, or each if it applies, each field can have a parameter for its home group, and one for any visitor. The following change to the chosen field inputs, f_A and f_B, extends this to handle uneven groups:

$$f_A = \frac{N_A}{N_A + N_B} \tag{7.20}$$

$$f_B = \frac{N_B}{N_A + N_B} \tag{7.21}$$

This is appealing because θ_{fg} then represents the worth of an average-rated individual from group g on field f.

Therefore, the field inputs are the relative size of each group. The full model including the field inputs then becomes:

$$Output = Pr(A \text{ defeats } B) = \frac{1}{1 + e^{-(w_A - w_B + \theta_{fA} f_A - \theta_{fB} f_B)}}. \tag{7.22}$$

The weight update after each comparison on field f then extends the delta-rule update from (7.11) to include the new parameters:

$$\Delta\theta_{fA} = \eta(1 - Output) f_A \tag{7.23}$$

$$\Delta\theta_{fB} = -\eta(1 - Output) f_B. \tag{7.24}$$

7.3.5 Taking Into Account Time

One of the appealing properties of viewing a Bradley-Terry model as an ANN is that adding extensions is often as simple as including an additional input into the ANN. Time can therefore be accounted for by adding inputs x_{t_A} and x_{t_B} with corresponding time weights θ_{t_A} and θ_{t_B} into the ANN. x_{t_A} is the time input for subject A and x_{t_B} is the time input for subject B. Again, subject A is the winning subject and subject B, the losing subject. Since the ANN is being used to learn the significance of a paired difference, x_{t_A} should be positive and x_{t_B} negative. The magnitude of the input can be encoded appropriately to the given data to be fit. For example, if there is a known maximum time limit the time input can be the percent of that maximum that has passed. If time is dependant on the particular "field" the subjects are being compared on, then a time weight can be learned per subject per field. If it can be assumed that all visiting competitors are affected equally by time on a given "home field", each field can have one weight for its home subject, and one for any visitor.

The time weight is updated in the same manner as the subject weights w_A and w_B by applying the delta rule:

$$\Delta\theta_{t_A} = \eta(1 - Output)x_{t_A} \tag{7.25}$$

$$\Delta\theta_{t_B} = -\eta(1 - Output)x_{t_B}. \tag{7.26}$$

The terms x_{t_A} and x_{t_B} are included here because they are not necessarily 1 or -1 as is the case with the subject inputs on weights w_A and w_B.

7.3.6 Rating Uncertainty

One of the problems of the given model is that when an individual participates for the first time, their rating is assumed to be average. This can result in incorrect predictions when a newer individual's true rating is actually significantly higher or lower than average. Therefore, it would be appropriate to include the concept of uncertainty or variance in an individuals rating. Glickman [28] derived both a likelihood-based method and a regular-updating method (like Elo's) for modeling this type of uncertainty in Bradley-Terry models. However, his methods did not account for finding individual ratings within groups. Instead of extending his models to do so, we give the following heuristic which models uncertainty without negatively impacting the gradient descent method.

Given g_i, the number of times that individual i has participated in a comparison, let the *certainty* γ_i of that individual be:

$$\gamma_i = \frac{min(g_i, G)}{G} \tag{7.27}$$

where G is a constant that represents the number of comparisons needed before individual i's rating is fully trusted. The overall certainty of the comparison is then the average certainty of the individuals from both groups. If a weighting is used,

then the certainty for the comparison is the weighted average of all the individuals in either group. The probability estimate then becomes:

$$Output = Pr(A \text{ defeats } B) = \frac{1}{1 + e^{-(c(w_A - w_B) + \theta_{fA} f_A - \theta_{fB} f_B)}}, \tag{7.28}$$

where c is the average certainty of all of the individuals participating in the comparison. This effectively stretches the logistic function, making the probability estimate more likely to tend towards 0.5—which is desirable in more uncertain comparisons. For example, assuming the map weights are close to 0, a comparison yielding a 70% chance of group A defeating group B with $c = 1.0$, would yield a 60% chance of group A winning if $c = 0.5$. The modified weight update appears as follows:

$$\forall i \in A, \Delta\theta_i = c\eta(1 - Output) \tag{7.29}$$

$$\forall i \in B, \Delta\theta_i = -c\eta(1 - Output). \tag{7.30}$$

Since this can also be seen to effectively lower the learning rate, η is in practice chosen to be twice as large for the individual updates as the field-group updates.

The down side to this method of modeling uncertainty is that is does not allow for a later change in an individuals rating due to long periods of inactivity or due to improving rating over time. This could be implemented with a form of certainty decay that lowered an individual's certainty value γ_i, over time. Also, notice a similar measure of uncertainty could be applied to the field-group parameters.

7.3.7 Preventing Rating Inflation

One of the problems with averaging (or summing) individual ratings is that there is no way to model a single individual's expected contribution based on their rating. An individual with a rating significantly higher or lower than a group they participate in will have an equal update to the rest of the group. This can result in inflating that

individual's rating if they constantly win with weaker groups. Likewise, a weaker individual participating in a highly-skilled but losing group may have their rating decreased farther than is appropriate because that weaker individual was not expected to help their group win as much as the higher rated individuals were. One heuristic to offset this problem is to use an individual's own rating instead of that individual's group rating when comparing that individual to the other group. This will result in individuals with ratings higher than their group's average receiving smaller updates than the rest of the group when their group wins, and larger negative updates when they lose. The opposite is true for individuals with ratings lower than their group's average. They will be larger when they win, and smaller when they lose. This attempts to account for situations where there are large differences in the expected worth of participating individuals. This substitution can be written as:

$$w_A = \theta_i \tag{7.31}$$

if individual i is in the winning group, and

$$w_B = \theta_i \tag{7.32}$$

if individual i is on the losing group. The substitution is only used to calculate $Pr(A \text{ defeats } B)$ for the *Output* used in player i's update.

7.4 Experiments

The final model employs all of the extensions discussed in section 7.3. The model was developed originally to rate players and predict the outcome of matches in the World War II-based online team first-person shooter computer game Enemy Territory. With hundreds of thousands of players over thousands of servers worldwide, Enemy

Territory is one of the most popular multi-player first-person shooters available. In this game, two teams of on average 4-20 players each, the Axis and the Allies, compete on a given "map". Both teams have objectives they need to accomplish on the map to defeat the other team. Whichever team accomplishes their objectives first is declared the winner of that map. Usually one of the teams has a time-based objective—they need to prevent the other team from accomplishing their objectives for a certain period of time. If they do so, they are the winner. In addition, the objectives for one team on a given map can be easier or harder than the other team's objectives. The size of either team can also be different and players can both enter, leave, and change teams at will during the play of the given map. These characteristics motivated the development of the above extensions because they are common in public Enemy Territory servers. Weighting individuals by time deals with players coming, going, and changing teams. Incorporating a group size-based "home field advantage" extension deals with both the uneven numbers and the difference in difficulty for either team on a given map. The time parameters allow the model to adapt to situation where the length of the match is unusual for the given teams. The certainty parameter was developed to deal with inflated probability estimates given several new and not fully tested players. Since it is common practice for a server to drop a player's rating information if they do not play on the server for an extended period of time, no decay in the player's certainty was deemed necessary.

The model was developed to predict the outcome for a given set of teams playing a match on a given map. Therefore, the individual and field-group parameters (in this case both an Axis and Allies parameter for each map) are updated after the winner is declared for each match.

In order to evaluate the effectiveness of the model, its extensions, and training method, data was collected from 3,379 competitions including 4,249 players and 24 unique maps. The data included which players participated in each map, how long

they played on each team, which team won the map, and how long the map lasted. Custom software was implemented in Python to simulate the ANN and its training. The ANN model is trained as described in section 7.3, using the update rule after every map. The model is evaluated five times, only adding the time parameters for the last run. The runs without the time parameters include one with no heuristics, one with only the certainty heuristic from section 7.3.6, one with only the rating inflation prevention heuristic from section 7.3.7, and one combining both heuristics. The final run includes all of the heuristics because they are shown to be effective, and it also includes the time parameters.

For the results given, the model's accuracy on a given match is always measured *before* being trained that match. This leave-one-out approach ensures the results are not biased in favor of the data used for training. It does, however, result in a bias that leads to less accurate results initially and then more accurate results near the end. This however, is appealing because it gives a measure of the performance of the model when used for the real-time application for which it was designed—namely continually ranking and rating players in this online game. When applied in the real-world, the model will be fit in this exact sequential manner, and therefore the results given are specifically appropriate to the application.

One way to measure the effectiveness of these runs would be to look at how often the team with the higher probability of winning does not actually win. However, this result can be misleading because it assumes a team with a 55% chance of winning should win 100% of the time—which is not desirable. The real question is whether or not a team given a $P\%$ chance of winning actually wins $P\%$ of the time. Therefore, the method chosen to judge the model's effectiveness uses a histogram to measure how often a team given a $P\%$ chance of winning actually wins. For the results in section 6.6, the size of the histogram intervals is chosen to be 1%. This measure is called the *prediction error* because it shows how far off the predicted outcome is from

Table 7.1: Bradley-Terry ANN Enemy Territory Prediction Errors

Heuristic	None	Inflation	Certainty	Both	With Time
Prediction Error	0.1034	0.0635	0.0600	0.0542	0.0457

the actual result. A prediction error of 5% means that when the model predicts team A has a 70% probability of winning, they really have a probability of winning between 65-75%. Or, in the histogram sense, it means that given all of the occurrences of the the model giving a team a 70% chance of winning, that team will actually win in 65%-75% of those occurrences.

As mentioned in section 7.1, there are no currently published models which with to compare this one, so none are given. It can only be said that given the complexity of the problem, producing reasonable estimates of match outcomes is, in and of itself, significant.

7.5 Results and Analysis

The results are shown in table 7.1. Each column gives the prediction error resulting from a different combination of heuristics. The first column uses no heuristics, the second uses only the inflation prevention heuristic (see section 7.3.7), the third column uses only the rating certainty heuristic (see section 7.3.6), the fourth column combines both heuristics, and the last column combines both heuristics along with taking into account the time parameters. Each heuristic improves the results and combining both gives a prediction error at 5.42%. Adding in the time parameters brings the error 16% lower to 0.0457. This value means that a given estimate is, on average, 4.57% too high or 4.57% too low. Therefore, when the model predicts a team has a 75% chance of winning, the actual average % chance of winning is between 70-80%.

One question is on which side does the prediction error tend for a given probability. To examine this, two plots are shown. Each plot gives the predictions plotted

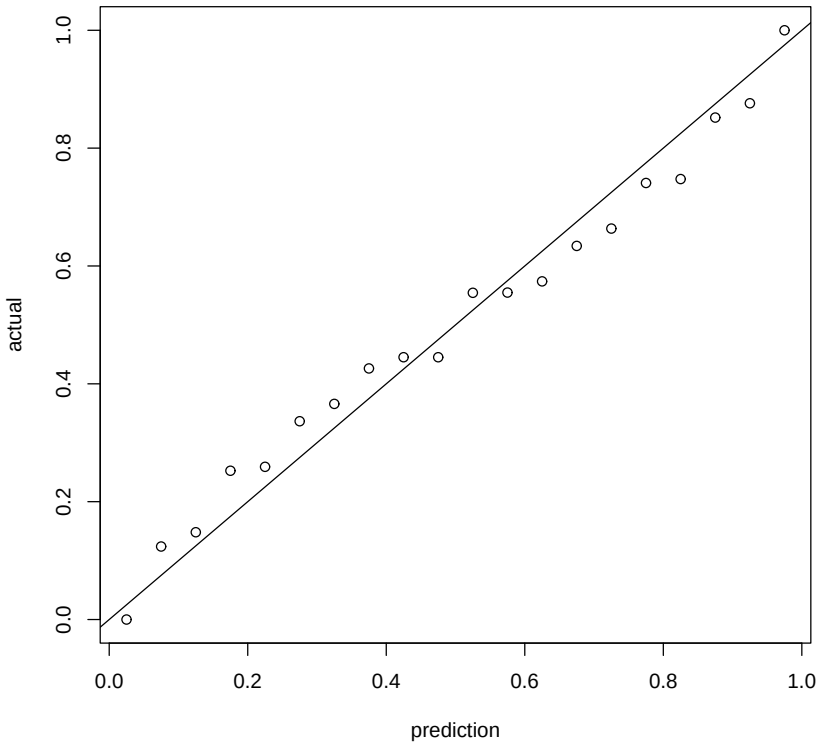

Figure 7.2: Prediction vs. true probability using 20 5% histogram intervals.

against the true values taking 20 5% histogram intervals of the results. Figure 7.2 shows the predictions for the final model without the time parameters, and figure 7.3 shows the predictions with the time parameters. The lines in the middle of each figure give the ideal, and therefore deviations from the line show the direction of the error for each prediction. Notice that with only two low-error exceptions on the extremes, the prediction error for the model without the time parameters tends to be positive when the prediction probability is low, and negative when it is high. This means that the original model is slightly extreme on average. For example, if the model predicts a team is 75% likely to win, they will be on average only 70% likely to win. A team predicted to be 25% likely to win will be on average 30% likely to win. The model with the time parameters, however, has the opposite trend. It

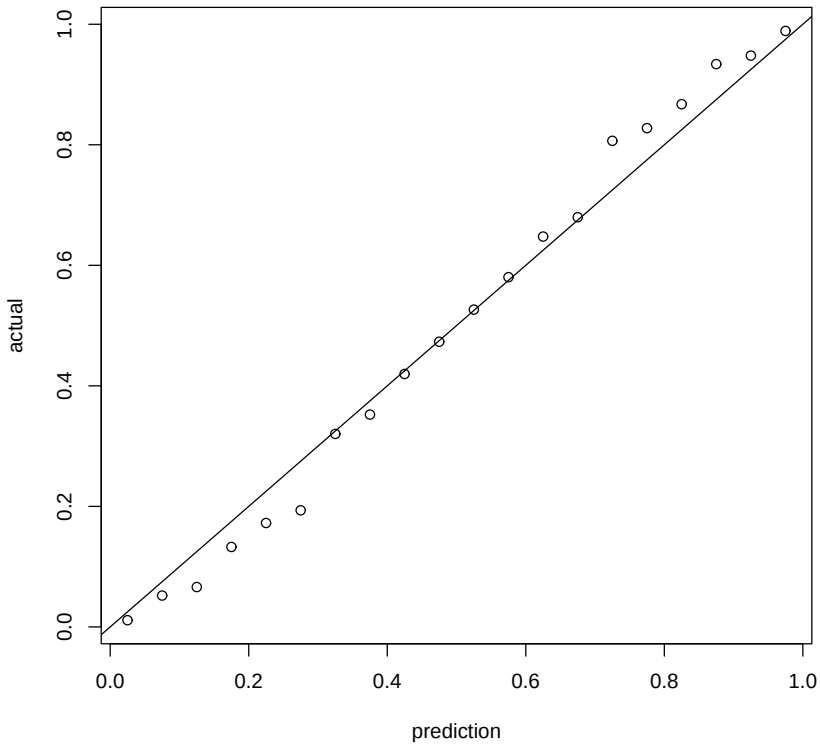

Figure 7.3: Prediction vs. true probability using 20 5% histogram intervals with the time-based parameters added.

tends to have negative error when the prediction is low, and positive when it is high. This means it is more conservative. When the newer model predicts a team is 75% likely to win, that team is really, on average, closer to 80% likely to win. When it predicts a team in 25% likely to win, that team is closer to 20% likely to win. It is not surprising that taking into account time results in a more conservative model because it lowers the expectation of a superior team winning if that team is "taking too long" to win. Another interesting difference when taking into account time is the fact it is more accurate near 50% (and therefore more likely to predict ties correctly) and the clusters are "tighter", meaning it has a lower variance.

In addition to evaluating the model analytically, the ratings it assigns to the players can be assessed from experience. For the matches used for the results, the well-known best players also have the highest ratings according to the model. These players are well-known on for their tenacity in winning maps, and therefore the model's ranking appears to fit intuition gained from experience with these players. In summary, the model effectively predicts the outcome of a given map with a prediction error of less than 5%, and provides reasonable ratings for the players.

7.6 Application

One of the goals in creating an entertaining multi-player game or running an entertaining multi-player game server is keeping the gameplay fair for both teams. Players are more likely to continue playing a game or continue playing on a given game server if the competition is neither too easy nor too hard. When the gameplay is uneven, players tend to become discouraged and leave the server or play different games. This is where the rating system becomes useful. Besides being able to rate individual players and predict the outcome of matches in a multiplayer game like Enemy Territory, the predictions can also be used during a given map to balance the teams. If a team's probability of winning is greater than some chosen threshold, a player can be moved

from that team to the other team in order to "even the odds." The Bradley-Terry ANN model as adapted for Enemy Territory is now being run on hundreds of Enemy Territory servers involving hundreds of thousands of players world-wide. A large number of those servers have enabled an option that keeps the teams balanced, and therefore, in theory, keeps the gameplay entertaining for the players.

One extension of this system would be to track player ratings across multiple servers world-wide with a master server and then recommend servers to players based on the difficulty level of each server. This would allow players to find matches for a given online game in which the gameplay felt "even" to them. This type of "difficulty matching" is already used in several online games today, but only for either one-on-one type matches, or teams where the players never change. The given Bradley-Terry ANN model could extend this to allow for dynamic teams on public servers that maintain an even balance of play.

This can also be applied to improving groups chosen for sports or military purposes. Individuals can be rated within groups as long as they are shuffled through several groups and then, over time, groups can be either chosen by using the highest rated individuals, or balanced by choosing a mix for each group. The higher-rated groups would be appropriate for real encounters, whereas balanced groups may be preferred for practicing or training purposes.

7.7 Weight Analyses

Besides the importance of the accuracy of the proposed model, also of interest are the values of the parameters themselves. Analyzing these parameters can lead to improvements in gameplay. For example, examining the time parameters and the field parameters can lead to insights that allow server administrators to more fairly balance matches, and allow level and map designers to either create more balanced maps, or recommend more appropriate time limits.

Table 7.2: Time and Field Weights for the baserace Map

	Field	Time
Allies	1.86	0.12
Axis	2.01	-0.12

Table 7.3: Time and Field Weights for the ice Map

	Field	Time
Allies	-0.97	1.04
Axis	1.78	-1.04

As an example, consider the time and field weights assigned to two maps shown in tables 7.2 and 7.3. First, note that the time weights are symmetrical. This is not surprising because the weight updates are the same for the time weights. In fact, both the field and time weights could be stored in a single weight, but they are separated for ease of interpretation. Notice in the baserace map (table 7.2) the field weights are close and the time weights are small. This exactly matches intuition about baserace because it is a perfectly symmetrical map. The objectives are the same for both sides, and the physical layout is also the same on both sides of the map. Therefore, the expectation is that the weights should be very similar, and time should not have a strong affect on the map. The ice map, on the other hand, is almost the opposite. It is far easier in practice for the Axis team to win the ice map—however when they do win, they usually win quite early. The negative time weight for the Axis team suggests that they become less and less likely to win as time passes. In practice, this usually means the Allies have found a way to defend against the Axis on this map more effectively than usual. Notice, however, that the difference in the field weights is still greater than the difference in the time weights. This suggests that even though the Axis team will become *less* likely to win over time, they are still expected to win more often in general. This also suggests that the model has detected that the ice map's timelimit may be too long because the input for the time weights is the

amount of time the map took over the timelimit. If shortened, the field and time parameters may come more into balance. This was seen using the team balancing system in practice.

As suggested above, the predictions from the Bradley-Terry model as applied to Enemy Territory can be used to actively balance the teams in a given match. In earlier tests, balancing the team without taking time into account resulted in maps like ice being heavily stacked in favor of the defense. This is because without the time weight parameter, the model can only assume that without a greater number of players, the Axis team will win most of the time. Adding the time weight parameter to this system results in more balanced play. It still stacks the defense initially, however, as time passes, the time-weighted model detects the Axis team is not as likely to win as originally predicted, and moves more players off the defense. Even with the time weighting, however, it was not uncommon for the team balancing system to not allow *any* players on the Axis team for several minutes. This is due to the aforementioned domination of the time weights by the field weights. Here, the team balancing system is effectively giving the unfavored Allies team a "head start" While logically correct, this is less entertaining for the players, who prefer to have contestants to play against, regardless of the difficulty. Fortunately, the "head start" the team balancing system suggests also implies that the map's timelimit may be too long. By decreasing the time limit of the map by roughly the amount of time the balancing system kept the teams empty, the time weight can come more into balance with the field weight. This results in a more balanced experience for both teams. Therefore, the results of adding time weights not only improves the model's predictive power, but also leads to new insights that improve gameplay overall. This suggests the model can be used not only for prediction, but map creators can use it for creating more balanced and therefore more entertaining maps. Games that provide a more entertaining experience are

generally more popular, which is beneficial to both the makers of the games, and the players who will have more people with which to participate.

Analyzing time weights can be used for more than just improving the balance of online games. It can be applied to any situation where determining the effects of time on a comparison or competition is important. In the military sense, it can be used to balance exercises or to determine how long to allow an encounter to continue. For sports it can be used to better determine how long a match should last. This can include team sports as well as other time involved matches like ice-skating where a skater has a set amount of time in their program. In education, it could be applied to how long it takes to learn a given concept when comparing different methods of teaching. In medicine, in can be applied to determine how time will affect the recovery or worsening of a given patient using different treatments. There are several significant applications where analyzing the effects of the passage of time can lead to new insights on important paired-comparisons.

7.8 Conclusions and Future Work

This paper has presented a method to learn Bradley-Terry models using an ANN. An ANN model is appealing because it makes using and extending the Bradley-Terry model accessible for a broader community. In addition, ANN re-parameterization also provides a training rule that can be used incrementally or iteratively. The basic model is extended to rate individuals where groups are being compared, to allow for weighting individuals, to model "home field advantages", to take into account the effects of time on gameplay, to deal with rating uncertainty, and to prevent rating inflation. The results when applied to a large, real-world problem including thousands of comparisons involving thousands of players yields probability prediction estimates within less than 5% of the actual values. In addition, analysis of the resulting parameters leads to new insights that suggest ways of improving the game's overall balance.

This type of individual rating out of groups can be important in many other applications. This can include rating individual players on sports teams, rating soldiers that participate in military group training, rating employees who participate in competing projects across different groups, rating competing manufacturing processes at the machine-level, and any other application where it is advantageous to rank individual subjects when they are only compared at group levels. In today's world, overall performance is measured more often than not at the group instead of individual level, and therefore approaches for rating in that context are becoming more and more important. The fact that the given method provided here performs well on an application as complex as public-style online gaming suggests it could perform well on other important applications as well.

One of the problems with the current model is that all associations are assumed to be additive and exponential. They are additive in the sense that the skill of a given group is proportional to the exponentiated sum of the skill of the individuals in that group. It does not take into account higher-level interactions between the individuals. It is exponential because the effects of time and the number of individuals per group is assumed to affect the model in an exponential fashion. This may be inaccurate if having more individuals implies more or less an exponential increase in the skill, or being in a group twice as long affects the outcome in a non-exponential fashion. Two approaches to improving the robustness of the current model include:

1. Extending the current single-layer ANN to a multi-layer ANN.
2. Constructing a higher-level statistical model that allows for these additional complexities.

One of the nice properties of using an ANN as a Bradley-Terry model is that it can be easily extended from a single-layer model to a multi-layer ANN. No currently known Bradley-Terry model incorporates interaction effects between individuals within a group or in a competing group. Current models do not consider that

individuals i and j may have a higher effective rating when playing together than apart. A multi-layer ANN is able to model these higher-order interactions by making each hidden node represent a subset of the stronger interactions. In addition, a multi-layer ANN is able to learn non-linear functions of its inputs, and therefore a multi-layer Bradley-Terry ANN can find if either or both of the group size and time inputs affect the outcome in a different fashion. The next Bradley-Terry model will be a multi-layer Bradley-Terry ANN constructed to determining these relationships.

One of the down sides of using a multi-layer ANN is that it will be harder to interpret the individual ratings. Therefore, statistical approaches including hierarchical Bayes will also be applied to see if a more identifiable model can be constructed. One way to develop an identifiable interaction model can extend from the fact that it is common for sub-groups of players to play in "fire-teams" or "squads" in games like Enemy Territory. An interaction model can be designed that uses the fire-teams as a basis without needing to consider the intractable number of all-possible interactions. For the time and group size effects, more robust, non-linear functions can be chosen and fit for combining these parameters. Using a statistical model would also make it more natural to account for the uncertainty in individual ratings. The uncertainty can be modeled directly as the variance in a given individual's rating. This more principled approach to handling uncertainty can then be used to replace the Bradley-Terry ANN model parameter c with an true estimate instead of a heuristic.

Chapter 8

Hierarchical Models for Estimating Individual Ratings from Group Competitions

Abstract

Providing direct and indirect contributions of more than $18 billion to the nation's gross output in 2004, the computer and video gaming industry is one of the fastest-growing sectors of entertainment. A large part of that market includes team-oriented online games. Players in these games often have a high-level of interest in statistics that help them assess their ability compared to other players. However few models exist that estimate individual player ratings from team competitions. There are models that can be used at the team-level, however the dynamic nature of the teams in the more popular public-style play of these games makes it necessary to build team strengths from player abilities. The following presents models that describe team abilities in terms of how well the individual players on the teams contribute to their team's winning. In addition, the models presented include parameters that estimate other characteristics of the games themselves. The models are posed in a hierarchical Bayesian framework. In addition to giving players a better estimate of their skill, these models can also be used to improve current gameplay, and create more enjoyable games in the future. Companies and servers that apply well-developed statistics for assessing their players' abilities are more likely to attract and retain players, leading to greater success in the industry.

8.1 Introduction

According to one marketing research firm, the computer gaming industry is the fastest-growing sector of entertainment [18]. Robert Crandall, a senior fellow with the Brookings Institution, and Gregory Sidak, visiting professor of Law at Georgetown University stated in a recent study that "The entertainment software business provided direct and indirect contributions of more than $18 billion to the nation's gross output in 2004" [13]. In fact, "video games represent an $11 billion U.S. entertainment industry, larger than Hollywood box office sales" [18]. A large portion of this revenue comes from the online gaming sector, which is projected to bring in $2.5 billion in 2006 [38]. 44% of all video and computer gaming players say they play online in addition to offline games [71]. One of the more popular genres in online gaming involves team competitions. This includes games based on popular sports, but more popular are team-based first-person shooter games which include blockbuster titles like Halo 2 and Half-Life 2. Millions of "gamers" can be found playing these games every day.

Most online players prefer having a method of tracking their progress in the games they play, and comparing themselves to other players. If a player feels like he or she can achieve some goal with a given statistic, they are more likely to continue playing. Games and servers that provide better developed statistical methods for comparing a player's skill are more likely to retain a larger player base, and hence be more profitable. Therefore, having good ways of rating players benefits the players, the companies producing the games, and the companies running servers for the games. In addition, if aspects of the game besides the players can be judged statistically, the developers can use this information to improve the quality of newer games. For example, in this paper we show how the fairness of the "field" two teams play on can be analyzed. This can be used to improve the fairness of gameplay.

Most players of online team games play on "public servers". By public we mean that players are free to join, leave, and switch teams as often as they like. By server, we refer to the software that maintains the state for a given game and match and determines when a match is over and who the victors are. Every match is highly dynamic in terms of which players are on either team and therefore the ratings of the teams can not be estimated at the group level, but need to be built from individual ratings. Unfortunately, for team-based games, there are few systems in place that estimate individual player ratings from group competitions. The following presents an extension of the model proposed by [7] for paired-comparisons that estimates individual player ratings based on the winners of team competitions.

There are several types of statistics that can be tracked in these team competitive online games, many of which are interesting at the individual level, but are not consistent indicators of a player's team contribution. The goal of the models in this paper is to estimate how well a player contributes to his or her team winning the match. Therefore, the ratings are estimated only in terms of the wins a player achieves while playing on their given teams. It should be noted that the ratings here are not necessarily strong indicators of how these players would play one-on-one, but rather in team settings.

There are several reasons why ranking and rating players in public, online, team-based competitions can lead to an increase in the number of players that play and continue to play a given game or play on a given server:

- It gives players a measure of their performance and hence motivation to improve themselves. Our experience with running several online servers has shown that players tend to play more often on servers that provide a method of assessing their performance. The servers providing more statistical information are consistently full whereas the others are often near empty.

- It allows server administrators to judge the fairness of gameplay on their servers. Servers known for fair gameplay always attract more players than other servers. This is because the newer players know they have just as much a chance of winning on either team, and older players know the challenge level will remain consistent.
- It can aid players in choosing servers that better fit their abilities. If the easiness of a server can also be judged, then players can choose the servers that best fit their abilities. The following paper provides a model for comparing the difficulty level of a given server to other servers.

In the end, the games and servers that provide better statistical measures in addition to more challenging and fair gameplay will attract more players.

The game chosen for this paper is called Wolfenstein: Enemy Territory, also known as ET. It is one of the most popular team-based online games played with several hundred thousand unique players playing every day. Its popularity is partly due to the fact that it was made free by its publisher, Activision. In Enemy Territory, each match consists of an Axis side and an Allies side competing on a World War II historically inspired map. Each team has several objectives they need to complete within a certain time limit in order to win. Usually, one team's objective is to accomplish a given goal, while the other team's objective is to prevent them from accomplishing this goal during the time limit. This often unbalanced play-style, combined with the common coming and going of players on a public server, creates matches that require rich models in order to predict. They can be compared to a soccer game where either team can have as many players as they like so long as the combined number of players on the field is less than some maximum. To complicate matters further, the players are free to switch between teams at will. Furthermore, the soccer field does not have to be symmetrical: the field can be on an incline such that the ball naturally rolls towards one team's goal, and it can also include obstacles

on one end that are not on the other. The pool of possible players also does not have a limit, ranging as high as 1,000, and the players are free to leave and join a match at any time. There are no other published models designed to handle this complex of a competition. The following presents a model to estimate individual ratings from team-play and account for both the imbalance in the play-style, and the dynamic nature of the teams.

With the models presented here, we are able to answer several questions, including, "Who are the best players on this server?", "Which maps are the most difficult for the Axis side?", and "Which server is easier to play on for the average player?", in addition to questions like "How likely are the Allies to win the current match on this server?"

This paper proceeds as follows. Section 8.1.1 gives related work, section 8.2 describes the data analyzed, the models are presented in section 8.3, section 8.4 explains how the model parameters are estimated, section 8.5 shows the results of the estimation, section 8.6 shows ways in which having the ratings and rankings can be applied, and section 8.7 gives conclusions and directions for future work.

8.1.1 Related work

Statistically motivated rating systems are not new. One of the earliest examples is that of [22] for chess ratings. He used the equivalent of a *Thurstone Case V* model which assumed the chess competitors had a rating score that followed a normal distribution. This model also assumed a normal distribution on the difference in player ratings. More recent versions of Elo's chess rating system have adopted a logistic distribution instead of a normal distribution because it has been shown that "weaker players have significantly greater winning chances" [73]. The logistic distribution version of Elo's model is equivalent to the model proposed by [7] for paired-comparisons.

The basic model used in this paper is an extension of the commonly used Bradley-Terry model. In this model, two opponents have ability parameters λ_1 and λ_2, and the probability of the first opponent winning is $\lambda_1/(\lambda_1 + \lambda_2)$. Several reviews on uses of and extensions to Bradley-Terry models can be found in the literature [7, 17, 16, 37]. One common extension proposed by [1] is to add a parameter to the model for home field advantage. The following uses a similar approach, but instead of a single parameter, there are two per map—one for the Axis side and one for the Allies side. In addition, [28, 26] recently extended the Bradley-Terry chess rating model to take into account uncertainty about a player's rating based on the amount of time that has passed since a player has competed.

What makes this work unique is that it presents a new model for estimating individual ratings from team competitions. Few other models have been presented for this situation. One notable model is that of [36]. They use Bradley and Terry's original formulation of the model, but obtain each team's λ by summing the ratings of its players: $\lambda_1 = \sum_{i \in 1} p_i$, where p_i is the rating of player i. This model fit well in their application which never had uneven amounts of individuals per group. It results in a linear increase in team rating when there are more players on one team than another. Experience has shown us, however, that having more players on one team does not result in a linear, but rather exponential increase in the team's perceived ability to win for our application. Another similarity between our model and that of [36] is that we both present a method for weighting individuals by how much they are expected to contribute to their respective teams. [36] suggest the weighting be predetermined using expert knowledge about the data, whereas we use a simple exposure model that uses the percent time a player plays on a given team for a given match as that player's "weight" for that match.

Another recently proposed model that estimates individual ratings from team competitions is that of [33]. Their ratings system is similar to the Thurstone Case

V model given by [22] except instead of comparing two individuals, it compares two groups by summing the ratings of the individuals. Since the likelihood they use is a normal, unlike the model given by [36], team ratings increase exponentially when one team has more players than another. The main difference in the model presented here is that we assume a logistic instead of normal distribution on the difference in team strengths. This results in our model being an extension of the Bradley-Terry model, whereas theirs is an extension of the Thurstone Case V model. We made this decision because we felt weaker teams were more likely to win than the normal distribution would predict.

8.2 Data

We obtained data by parsing the log files from three Enemy Territory servers yielding around 100 matches per server. For each match, the log files list which server the match was played on, which map was played, which side won—Axis or Allies—the length of the match, which players participated, and how long each player spent on both teams. Here is an example of a log file entry from one match:

```
InitGame: ...\mapname\fueldump\...
Winner: AXIS Time: 1800000
Name: |R!P|Orpheo GUID DFBB5: Time Axis: 0 Time Allies: 1450200
Name: |R!P|Crazyeskimo GUID EF071: Time Axis: 1549800 Time Allies: 0
Name: sliveR GUID 0A589: Time Axis: 1614950 Time Allies: 0
Name: DaSaNi GUID 3F6C7: Time Axis: 1278400 Time Allies: 0
Name: <ETA>Ronik GUID DF58D: Time Axis: 1446150 Time Allies: 0
Name: pXo GUID 67F24: Time Axis: 0 Time Allies: 1491900
Name: -|S*S|-Chewie GUID 89E44: Time Axis: 1364050 Time Allies: 0
Name: [MIST]Lars GUID 54ECD: Time Axis: 1584700 Time Allies: 0
```

```
Name: SexY SpenceR GUID 828D1: Time Axis: 0 Time Allies: 1444250
Name: Gunny Highway GUID B560E: Time Axis: 0 Time Allies: 1541850
Name: <ETA>Gorm GUID C66B6: Time Axis: 0 Time Allies: 1470750
Name: livefastdieyoung GUID 0E0FA: Time Axis: 1527650 Time Allies: 0
Name: MAC 10 GUID AB76C: Time Axis: 0 Time Allies: 1482550
Name: Sexy hamodi GUID 90838: Time Axis: 510200 Time Allies: 0
Name: xbigphillx GUID 265DE: Time Axis: 0 Time Allies: 877700
Name:  Alone GUID F48ED: Time Axis: 0 Time Allies: 437900
Name: -=AoW=- Red GUID 566FC: Time Axis: 0 Time Allies: 0
Name: Boas7 GUID 1EF51: Time Axis: 0 Time Allies: 0
Name: Eryx GUID 77068: Time Axis: 0 Time Allies: 0
Name: Monoxide GUID 8D4BD: Time Axis: 0 Time Allies: 0
Name: ProfessorX GUID 70D81: Time Axis: 0 Time Allies: 169950
```

All but the map name information has been removed from the first line of the log file entry because only that information is currently used. The next line, or "Winner" line gives the side that won the match and how long the match lasted in milliseconds. In this match, the Axis side won and the match lasted 1,800,000 milliseconds, or 30 minutes. The remaining "Player" lines give information about each player. First, a name is given. These are nick names the players choose and often change. In order to track players despite changing names, the next field is used. A player's GUID or *Globally Unique IDentifier* is a 32-digit hexadecimal field that is unique across all servers for Enemy Territory and assigned by a central server for every player everywhere in the world. The GUIDs presented here have been shortened to 5 digits for both privacy and brevity. The player records are stored based on this instead of their names which can change. After the GUID, the last two fields give the amount of time each player spent on the Axis or Allies team in milliseconds. For example, |R!P|Orpheo, the first given player, spent 1,450,200 milliseconds, or 24 of the matches'

30 minutes on the Allies side, which lost. |R!P|Crazyeskimo, the second player from the top, spent 1,549,800 milliseconds, or almost 26 of the matches' 30 minutes on the winning Axis side. Notice that neither player spent the entire 30 minutes of the match in the game. In addition, here is an example of a player in a different match who spent time on both sides:

```
Name: BlackSheep GUID 6C875: Time Axis: 552600 Time Allies: 1336200
```

Here, BlackSheep spends 9 minutes on the Axis team, and then 22 minutes on Allies team. It is not uncommon to see players that do not play the entire match on the same team and players who do not participate for the entire match in online matches like those in Enemy Territory. This happens for several reasons, including players joining the server late in a match, and players switching teams to play with closer acquaintances. Therefore, we present a method for handling this dynamic behavior in section 8.3.

8.3 Models

The following shows how we used the given data to estimate player ratings. We first present models for the individual player ratings, and then the likelihood for predicting whether a given team will win a match given the data above. We give three potential models for individual player ratings: a basic one in section 8.3.1, one that accounts for map-side effects in section 8.3.2, and a model that accounts for the difficulty of a given server in section 8.3.3.

8.3.1 Basic Model

Although the outcome of each competition is measured at the team- or side-level, the fundamental unit of the rating system needs to be built on the abilities of the individual players themselves. The likelihood given in section 8.3.4 determines a

team's rating from the players. Therefore, we need a model for the individual player ratings. Initially, we let θ_i represent player i's ability to help their side win a match. There are no existing standards for rating players in these games and so for simplicity we chose a model that places player ratings symmetrically around 0. The better players will have positive player ratings, and the worse players negative ratings. This is in contrast to the ratings in, for example, chess, that have values ranging closer to 1500–2500. Our first model for player ability is:

$$\theta_i \sim N(\mu, \sigma_\theta^2). \tag{8.1}$$

σ_θ^2 is given a prior distribution and $\mu = 0$ without loss of generality. This model makes the naive assumption that a player's probability of winning will not be affected by the difficulty of the side that player plays on for a given map.

8.3.2 Accounting for Map-Side Effects

Because maps in Enemy Territory and most other games of this nature are so varied, it is convenient to describe the details of a given match in terms of the combination of map and side that each player played on. Throughout this paper, we will refer to a team or a player's "map-side" combination, meaning the map the match took place on, e.g. the map "venice", and the side the player played on—Axis or Allies. The model given above is naive considering most maps were designed such that one side has a major advantage. Since this imbalance is uniform by design for all players, a more robust model is to assume that each player's ability increases or decreases according to a map-side rating. In this model, we let $\theta_{i,m-s}$ represent player i's ability to help side s win a match played on map m. As with player ratings, we choose a model that places map-side ratings symmetrically around 0:

$$\theta_{i,m-s} \equiv \theta_i + \gamma_{m-s} \text{ with } \theta_i \sim N(0, \sigma_\theta^2) \text{ and } \gamma_i \sim N(0, \sigma_\gamma^2). \tag{8.2}$$

This is similar to fitting the "homefield advantage parameter" proposed by [1], except that instead of one parameter per "field", here there are two—one for the Axis side, and one for the Allies. This is because it is not clear from the data which side has the advantage, and so we infer this by fitting a parameter for both sides instead of choosing a single side. We felt it easier to interpret each side's map rating separately, rather than have a rating for just one side. Although this does not model individual player-map-side interactions, it is assumed that the additive effect will be stronger than the individual interaction effects because most maps have been designed to be uniformly uneven. We are more interested in increasing the predictive power of the ratings than in the interactions themselves, and therefore we chose not to model them at this point.

With this model, a player's ability on a current map becomes his base rating, θ_i, offset by a map-side effect, $\gamma_{m,s}$. This information is interesting because it suggests that a team of skilled players can still enjoy a reasonable level of challenge against a team of less-skilled players as long as the map-side effects give the less skilled team an equivalent advantage. In addition, the estimates for $\gamma_{m,s}$ can be used to judge which maps are more balanced between Axis and Allies. Server administrators can use this information to choose which maps to place on their servers to improve gameplay, and map creators can use this information to better balance the gameplay in their maps.

8.3.3 Server Difficulty

If players can be ranked based on matches they participated in on a given server, it would be beneficial to also be able to compare players participating on different servers, or players that played on several servers. In order to determine how a given server affects a player's rating, the following model adds a server bias, ψ_j, into each

player's rating, yielding:

$$\theta_{i,m-s,j} \equiv \theta_i + \gamma_{m-s} + \psi_j \text{ with } \theta_i \sim N(0, \sigma_\theta^2), \gamma_i \sim N(0, \sigma_\gamma^2) \text{ and } \psi_j \sim N(0, \sigma_\psi^2). \quad (8.3)$$

Note that the server difficulty is modeled as an increase instead of decrease a player's rating. This means that we are really modeling server "easiness." We did this for simplicity in the model. It just means that lower and not higher values of ψ denote a more difficult server. With this model, a player's rating depends on their base ability, the side and map they are playing on, and the server they are playing on. Besides being able to compare players across servers more effectively, estimating ψ also gives us a measure of how difficult each server is. This information can be used by newer players to choose easier servers, or more experienced players can use it to find challenging gameplay. In addition, since the player ratings can now be fit globally over all servers, a player can compare his or herself to all of the players in the game, instead of only those who play on a certain server. The accuracy of the comparison, however, will be affected by how often players move between servers. If enough server "cross-over" occurs, then the ratings should be reasonable enough for comparison.

Another interesting aspect of ψ is since the amount added to each side is equal to $N_s\psi$ where N is the number of players on side s, ψ can be interpreted as the rating increase a given side s expects for each additional player on that side. Servers where having additional players will not make up for the skill of the players are more difficult than those where a few extra players alone can decide the winner. Since this additional information is interesting for rating a server alone, ψ is also fit in the single-server results given in section 8.5.

8.3.4 Likelihood

The likelihood we use is based on a modified version of the Bradley-Terry paired-comparison model [7]. In order to predict which of two teams is more likely to win, we choose a side s's probability of winning a match played on map m to be proportional to $\lambda_{s,m} = exp(\sum_{i=1,i\in P_s}^{|P_s|} \theta_{i,m-s,j})$ where P_s is the set of players on side s and $|P_s|$ is the number of players on side s. Therefore the probability of side s_{Axis} defeating side s_{Allies} for the given map is $\lambda_{s_{Axis},m}/(\lambda_{s_{Axis},m} + \lambda_{s_{Allies},m})$. This results in the probability of a given side winning increasing exponentially with the size of that side compared to the other. This is appealing because unlike a game like chess where only one move can be made per side despite the number of actual players making the move decisions, in Enemy Territory and other real-time online team competitions, every player can act simultaneously. Having more players gives a team with more numbers the ability to more effectively defend or accomplish its objectives. It is rare for a team with two or less players than another to win unless that team is very good.

If G is the total number of matches, $w(g)$ gives the winning side for match g, $l(g)$ the losing side, and m the map played on, then this model gives rise to the following likelihood function:

$$P(w|\lambda) = \prod_{g=1}^{G} \lambda_{w(g),m}(\lambda_{w(g),m} + \lambda_{l(g),m})^{-1} \tag{8.4}$$

where

$$\lambda_{w(g),m} = exp(\sum_{i=1,i\in P_{w(g)}}^{|P_{w(g)}|} (\theta_{i,m-s,j})) \tag{8.5}$$

and

$$\lambda_{l(g),m} = exp(\sum_{i=1,i\in P_{l(g)}}^{|P_{l(g)}|} (\theta_{i,m-s,j})). \tag{8.6}$$

One problem with the above likelihood is that in public-style servers players can come, leave, and change teams almost at will. Therefore, a model that does not

take into account how much time a player spends on each team is inaccurate. Since the match data we are using does indicate the amount of time the players spent per team, this information can be used to modify the λs above:

$$\lambda_{w(g),m} = exp(\sum_{i=1,i\in P_{w(g)}}^{|P_{w(g)}|} (\tau_{i,w(g)}\theta_{i,m-s,j})) \quad (8.7)$$

$$\lambda_{l(g),m} = exp(\sum_{i=1,i\in P_{l(g)}}^{|P_{l(g)}|} (\tau_{i,l(g)}\theta_{i,m-s,j})) \quad (8.8)$$

where $\tau_{i,w(g)}$ ($\tau_{i,l(g)}$) is the percent of the total match time player i spent on the winning (losing) team, $w(g)$ ($l(g)$). This yields a simple approximation to integrating the player abilities over the time of the match to estimate the overall strengths λ per team.

8.4 Analysis strategies

This section will discuss how the priors were chosen, how the model was fit, and how the convergence of the fit was verified.

8.4.1 Prior selection

The models presented here will have three priors. There is a prior on the standard deviation of the player ratings, σ_θ^2, a prior on the standard deviation of the map-side effects σ_γ^2, and a prior on the server effects σ_ψ^2. Instead of choosing non-informative priors for these standard deviations, we placed hyperprior distributions on them using inverse gamma distributions, with α and β chosen such that the distributions have means of 1.0 and variances at 1/3. This was done simply to keep most player ratings between -3 and 3. Again, we chose priors this way because there are no current standards for rating players in these types of competitions, and so we decided for

simplicity to assume they follow close to a standard normal distribution. We used hyperpriors so we could observe the relative differences in the standard deviations.

8.4.2 Software

Software to fit the models used in this paper was written in Python specifically for these analyses. It uses MCMC with Gibbs steps to sample from the standard deviations and Metropolis steps to sample from the player rating θs, the map-side effect γs, and the server difficulty ψs. This leaves a potentially large set of tuning parameters, but following the suggestion of [30], we chose separate constants for both player- and map-step sizes and divided them by the square root of the number of games a player had played in, or the number of times that map had been played. This resulted in acceptance rates of around 40% for the parameters fit using Metropolis steps.

8.4.3 Convergence Diagnostics

While it is difficult to guarantee an assessment of the mixing and convergence for MCMC, time series plots and acceptance rate analyses were used and the criteria in the diagnostics proposed by [62] were met. In addition, section 8.5.4 gives the results of using the Bayesian χ^2 goodness-of-fit test recommended by [39] on each run of MCMC used.

8.5 Results

In this section we derive rankings for players based on match information on three different Enemy Territory servers. First, rankings are derived for each server separately, and then all three are fit simultaneously. Combining the servers allows us to rank the players globally, and allows us to compare the difficulty of the three servers

using ψ. In addition, the difficulty of the maps with respect to which side a player chooses are analyzed.

8.5.1 Separate Server Rankings

In this section we present the top ten rankings of the players on each server separately. The model was fit once per server using matches from only that server. Each time, 100,000 iterations of MCMC were used with 10,000 iterations of burn-in. The results are shown in tables 8.1–8.3. The players are conservatively ranked by how they play two standard deviations below their posterior means. This has the desired effect of penalizing players whose ratings are less certain. It is not uncommon in these tables to see players who have worse winning percentages ranked higher than those with better ones. For instance, in table 8.1, BlackSheep has a higher posterior mean rating and is ranked higher than 0c3DiGiTaL despite having an 8-1 record vs. 0c3DiGiTaL's 9-0 record. This suggests that BlackSheep's wins were against more difficult odds—e.g. harder map-side combinations, better players, etc.

Since the players who play commonly on public servers rarely receive popular attention, the aliases used will not likely be recognizable to the general public. However, the goals of giving each player a rating and hence a stronger incentive to continue playing and improving his or her abilities in the game, in addition to better predicting the outcome of the games, have been met. That said, the players shown are known to be highly skilled amongst those who play on the servers.

8.5.2 Combining all 3 Servers

To determine which were the best players over all three servers, we took advantage of the fact that many players play on more than a single server to fit the server difficulty parameter, ψ, across all three servers simultaneously. Again, the model parameters

Table 8.1: Ratings and rankings for server 1. Shown are the player ranks, the posterior means of their base ratings, θ_i, the standard deviation of each player's θ_i, the nicknames they use in-game, the number of games each played, the number of times they won, and the number of times they lost.

Rank	Post. Mean	SD	Name	Games	Wins	Losses
1	1.90	1.02	K4F*Boas7	39	26	13
2	1.44	1.10	|R!P|Burnz	27	21	6
3	1.73	1.32	BlackSheep	9	8	1
4	0.73	0.89	|R!P|Crazyeskimo	52	30	22
5	0.98	1.07	SexY SpenceR	27	17	10
6	1.37	1.29	Ignis	7	7	0
7	1.29	1.29	—R!P—HunTeR	7	6	1
8	1.24	1.28	0c3DiGiTaL	9	9	0
9	0.66	1.09	=MG=.:Guen:.	26	15	11
10	0.46	1.01	-|S*S|-dgifTy!	28	17	11

Table 8.2: Ratings and rankings for server 2.

Rank	Post. Mean	SD	Name	Games	Wins	Losses
1	3.44	1.98	Sambuku	25	18	7
2	1.61	1.55	UruhA	32	22	10
3	1.91	1.70	-=DM=- nixx(cz)	18	13	5
4	1.56	1.61	<ETK>Soldier!Ryan	27	16	11
5	1.97	1.84	K4F*SHERREMAN	11	9	2
6	1.68	1.71	LaMigra	23	16	7
7	1.81	1.88	SMG.Goku.	17	14	3
8	1.77	1.93	PaRaDiCe	14	11	3
9	1.76	1.95	[BerserkS]FirebalL	8	8	0
10	1.72	1.94	[TKF]freezer	4	4	0

Table 8.3: Ratings and rankings for server 3.

Rank	Post. Mean	SD	Name	Games	Wins	Losses
1	1.70	1.05	b!t	52	32	20
2	1.74	1.11	SeXyBloodLust	42	26	16
3	1.99	1.34	Pizza Toby	11	10	1
4	1.38	1.28	Sexy* Wikked Man	18	14	4
5	1.40	1.30	Sexy Hu$tle	14	11	3
6	1.29	1.28	Smaug	18	13	5
7	0.75	1.01	SEXY SQUIDDY	46	25	21
8	0.87	1.106	BobbaloUSA	28	18	10
9	1.65	1.47	Assassin	23	14	9
10	1.32	1.43 BuzzKi11	7	6	1	

were estimated using 100,000 iterations of MCMC with 10,000 iterations of burn-in. The results are shown in tables 8.4 and 8.5.

The server results are surprising at first because most players consider the settings on server 1 to make it a more difficult server. However, the fact that the matches on server 1 are generally smaller because it is the least popular of the servers may contribute to it being easier overall for those players who play on multiple servers. It is not as surprising to see server 2 above server 3 since server 2 is the most popular server and attracts many of the best players. It is interesting the number of top players that do not come from server 2's top ten list, considering it should be the harder server. This suggests those that are in the top performed exceptionally across all servers despite the easiness of the servers.

Here we see an example where players could use this information to choose a server. The better players could choose to play on the 2nd server to enjoy more of a challenge, whereas newer players may be likely to choose server 3 because it is easier.

Table 8.4: Ratings and rankings for all three servers combined.

Rank	Post. Mean	SD	Name	Games	Wins	Losses
1	3.12	1.33	K4F*Boas7	66	43	23
2	2.97	1.58	SeXyBloodLust	42	26	16
3	3.22	1.72	|R!P|Burnz	27	21	6
4	3.81	2.04	Sambuku	25	18	7
5	2.04	1.20	b!t	74	41	33
6	3.39	1.98	Pizza Toby	11	10	1
7	3.47	2.05	BlackSheep	9	8	1
8	3.37	2.05	|R!P|HunTeR	15	12	3
9	1.83	1.48	UruhA	45	30	15
10	1.25	1.22	|R!P|Crazyeskimo	70	41	29

Table 8.5: This table shows the servers ranked by difficulty, giving the most difficult first. Also shown for each server is the posterior mean, the standard deviation, and which server is in question. Recall that smaller not larger values denote more difficult servers.

Rank	Post. Mean	SD	Server
1	1.48	0.75	2
2	1.99	0.82	3
3	3.04	0.90	1

8.5.3 Map-Side Effects

One of the interesting effects of the model used is that rating parameters are also fit for each map-side combination. This information can be used to judge which maps are more even and which maps have the least balance between Axis and Allies. Tables 8.6–8.9 show the results of fitting the map-side parameters on each server for first the most uneven map on that server, and then the most even map. In both cases, maps that are not common to all three servers were not considered. In general, the results are in line with the perception of the players. Both the *oasis* and *radar* maps are known to be difficult for the Axis side to "defend", whereas *fueldump*, *venice*, and *goldrush* are known for their fairness and commonly listed as player favorites. It is interesting, however, that the trend for the oasis map appears opposite on the third server from the other two servers and the global fit. This suggests there may be a fundamental difference in how that particular map is played on server 3, and suggests a possible need for examining map-server interactions.

From these findings, it is not unreasonable to conclude that, for example, oasis is imbalanced in favor of the Allies and that venice is probably a fair map. Server administrators could decide to either not include maps like oasis in the list of maps on their server, or they can take measures to ensure that when oasis is played, the Axis team consists of more skilled players. Map makers can use this information to consider whether they should make changes to oasis that would make it more even given equally skilled teams. Games with statistically driven level design are more likely to attract players, as are servers that include more balanced maps in their lineups.

8.5.4 Measuring Performance

The performance of the models was measured in two ways. First, by their accuracy in predicting the matches used to estimate the model parameters. This results in a

Table 8.6: Map-side ratings for first the most uneven map and then the most even map on server 1. The rows show the in-game name of the map, the side of the map described by the current row, the posterior mean of the map's rating or γ for that side, and the standard deviation of that map's rating.

Name	Side	Post. Mean	SD
oasis	Allies	0.50	0.44
oasis	Axis	-0.51	0.51
fueldump	Allies	0.08	0.30
fueldump	Axis	-0.37	0.33

Table 8.7: Map-side ratings for first the most uneven map and then the most even map on server 2.

Name	Side	Post. Mean	SD
oasis	Allies	0.90	0.42
oasis	Axis	-0.41	0.47
venice	Allies	0.03	0.25
venice	Axis	-0.08	0.26

Table 8.8: Map-side ratings for first the most uneven map and then the most even map on server 3.

Name	Side	Post. Mean	SD
radar	Allies	0.20	0.34
radar	Axis	-0.42	0.38
oasis	Allies	-0.09	0.25
oasis	Axis	0.02	0.25

Table 8.9: Map-side ratings for first the most uneven map and then the most even map for all three servers combined.

Name	Side	Post. Mean	SD
oasis	Allies	0.25	0.20
oasis	Axis	-0.34	0.23
venice	Allies	0.04	0.17
venice	Axis	-0.13	0.19

positive bias for the results. An unbiased estimate of the models' accuracy would require computationally intensive methods such as leave-one-out cross-validation. These methods would be impossible in practice because the amount of time it takes to fit the model using MCMC is longer than the length of the average match. Therefore, a future method will be developed that approximates the given models more efficiently. That said, each set of parameters achieved near 100% accuracy in predicting the match data. This suggests that the ratings derived for the players are reasonably accurate over the given matches.

For the second method for measuring model performance, we give the results of using the Bayesian χ^2 goodness-of-fit test recommended by [39]. The random variable in question is the discrete win or loss outcome of a match. [39] proves that "if $f(y|\theta)$ denotes the probability mass function of a discrete random variable y and

$$p_k(\tilde{\theta}) = \frac{1}{n}\sum_{j=1}^{n}\sum_{y\in \text{bin } k} f_j(y|\tilde{\theta}) \tag{8.9}$$

then the χ^2 statistic R^B may be defined as

$$R^B = \sum_{k=1}^{K}\left[\frac{(m_k - np_k(\tilde{\theta}))}{\sqrt{np_k(\tilde{\theta})}}\right]^2 ." \tag{8.10}$$

In our case we have 2 bins ($K = 2$) and therefore 1 degree of freedom. The bins consist of the wins and the losses, however the data only reports wins and therefore the losses bin is empty. Since one bin is empty, we only need to consider when $k = wins$. Therefore, for p_{wins}, $f(y|\tilde{\theta})$ is the probability the favorite should have won a given match for a single MCMC iteration's parameters or $\tilde{\theta}$. Given n is the number of matches, p_k is the average probability of the predicted favorite winning. m_{wins} is the number of data in bin $wins$, and since the data is reported in terms of wins, m_{wins} is also simply the number of matches. The resulting χ^2 statistic is therefore

Table 8.10: Bayesian χ^2 goodness-of-fit test results for each server and the combined model. Each row gives the name of the server in question and the average p-value of the χ^2 statistic over all 90,000 iterations of MCMC for that server.

Server	Mean p-value
1	0.02
2	0.17
3	0.04
All	0.01

proportional to the difference between the actual number of wins and the expected number of wins given the $\tilde{\theta}$ of one MCMC iteration.

Table 8.10 gives the results of the χ^2 test for each server individually, and for the model that combines all three servers. The second column of the table gives, for each server and for the servers combined, the average p-value over the 90,000 iterations of MCMC used to fit the models. For servers 1 and 3, and also for the model that combines all three, there is evidence that the model has fit well. However the evidence for the second server is not as conclusive. This may be due to the more varied nature of the players on the second server. The first and third server tend to entertain a more consistent and loyal crowd, whereas the second server is more likely to host players that stay for fewer matches. Since combining the three servers leads to a better fit overall, it is likely that having cross-server information about players from server 2 made up for that server's weaker fit.

8.6 Applications

This section describes potential applications for the suggested model.

8.6.1 Ranking the Players

Our first application is the obvious one, using the ratings to rank the players on the servers and across the servers. Players tend to prefer servers that give them a ranking they can work to improve. We note, however, that the method used to estimate the model parameters in this paper, MCMC, is too computationally intensive to be used real-time in situations involving millions of players on thousands of servers. This is because the amount of time it takes MCMC to fit the data is usually longer than the length of a single match. Therefore, we will develop a more efficient method for determining player, map, and server ratings. This method needs to update the model parameters quickly after every competition on every server. One of the benefits of having a real-time approximation to these models is that the information can be used in real-time to improve the fairness of the matches and, by fitting ψ, it can be used to help players choose servers.

8.6.2 Choosing Servers

Because we model server difficulty as well as player and map-side ratings, players can compare themselves across servers when they are fit simultaneously. This assumes there are enough players that play on more than one server to make comparisons meaningful. For example, if *none* of the players who play on the third server ever play on any other server, their ratings and their server's difficulty rating can not be correctly compared to other players and servers. Games that have this information can make it available to players who are trying to choose which servers to play on. Newer players can choose the easier servers, and veterans can choose servers that will give them more of a challenge. Also, because player ratings are fit taking into account the difficulty of the servers they play on, the servers can be also be listed in order of the average player ratings of the players *currently* playing. This measure gives an even better estimate of the current state of the server because it represents skill of

the players in the current match, instead of the average difficulty. Players can sort the servers by the average rating of the players on them, and then choose servers with ratings to suit the desired challenge level. As a more advanced option, games can be designed to automatically choose servers that best fit a given player. This can ensure that players always have a positive experience in playing these complex team-based games.

Besides a more consistent and balanced gameplay experience, giving players the ability to choose servers based on difficulty gives them another incentive to continue playing. Players will start on easier servers with the goal of improving their skills to a point they can comfortably play on harder servers. This natural improvement path will encourage players to return again and again to a game to see if they are good enough to play on harder and harder servers. One of the most popular online game types is the MMORPG or Massively Multiplayer Online Role-Playing games. The most popular of these games, World of Warcraft, boasts a player base of more than 6 million [75]. The draw to the MMORPG is that the game is designed with explicit character development paths that give players incentives to continue playing. However, these paths can be very time-consuming to pursue, especially for the more casual gamers that play in games like Enemy Territory. The server rating system can supply a character development path for the casual gamer based solely on the ratings system. This can enlarge the audience for first-person shooters and encourage players to buy games with this feature, and continue playing them.

8.6.3 Balancing Teams

At the server level, administrators can use player rating information to balance the teams currently playing. To make this easier, the game can use the likelihood to give posterior prediction estimates on which team is likely to win. If a team is very unlikely to win the match based on this information, administrators can move better players

to the disadvantaged team, and / or move worse players to the favored team. This can also be done automatically. If a team falls below a specified probability of winning, say 30%, then the game can automatically move players to bring the probability as close to 50% as possible. A computer can easily try all possible moves involving one player and in such a way greedily optimize the probabilities around 50%. Servers that employ automatic team balancing will enjoy a consistently balanced level of gameplay and attract more players. Games that have this option available may be more popular than those that do not.

In addition to making the game more enjoyable for its players, balancing teams in this manner can improve the efficiency of rating the players. As [33] observed, "the [team balancing] process can be viewed as a process of sequential experimental design" [24]. This is appealing and "since the quality of a match is determined by the unpredictability of its outcome, the goals of [balancing teams] and finding the most informative matches are aligned!" [33]. If the data used to fit the model comes from matches where a form of team balancing has been employed, less data may be needed to achieve a more accurate fit of the model.

8.7 Conclusions and Future Work

We have presented new models for estimating individual ratings in team competitions. These ratings allow players to effectively track their ability to help the teams they play on win. Games and servers that provide well-developed methods for tracking their players' abilities are more likely to attract players and therefore be more profitable. The models presented are also able to account for both the dynamic nature of the teams in public, online team-competitions, and the imbalance commonly associated with the levels or maps designed for these competitions. In addition, we have estimated parameters that allow us to analyze the fairness of the maps in terms of the sides playing. This information can be use to create maps that are more fair

and therefore more enjoyable for the players, or the information can be used by server administrators to choose different maps for their respective servers.

In addition, we have fit a parameter that allows us to estimate the difficulty of each of a set of servers. This information can be used to allow players to select servers that better match their abilities. Players are more likely to continue playing a game if they can ensure matches will not be too easy or too difficult. The models developed in this paper can be directly applied to modern-day online team-competitions with results that will improve gameplay and attract players.

We also note that the method used to estimate the model parameters in this paper, MCMC, is too computationally intensive to be used real-time in situations involving millions of players on thousands of servers. Therefore, we will develop a more efficient method for determining player, map, and server ratings. This method needs to update the model parameters quickly after every competition on every server. One of the benefits of having a real-time approximation to these models is that the information can be used in real-time to improve the fairness of the matches and for choosing servers to play on.

In addition to automatically balancing the teams in a current match, a faster approximation to these models can provide real-time information about the easiness of the servers in a game, or even a real-time estimate of the probability a given player is better or worse than the players currently playing on each server. Players can use this information in-game to select servers that better match their abilities.

Future work will also investigate additional attributes of the team competitions that may be interesting to analyze. For instance, a model similar to that proposed by [26] could be used to track time-varying player strengths. Some players are known veterans whose abilities are not likely to change appreciably over time, however others are newer players whose abilities are quickly improving. Modeling these changes can result in more accurate estimates of the players' ratings and therefore better

predictions of game outcomes. It could also show which players are improving the most over time.

It would also be interesting to model the effect of the number of players on how hard a given map-side combination is. There may be maps that are easier for a given side when there are fewer players, but become harder as the total number of players increases. In this model, γ_{m-s} would be the result of a regression fit instead of a single, learned constant. Being able to analyze the effects of the number of players on a maps fairness will allow server administrators to choose maps more appropriate for the number of players commonly on their servers, again leading to increases in server usage.

One way to efficiently estimate the parameters of the models given here is to design the aforementioned faster approximation as a recursive algorithm. A benefit of a recursively updating algorithm is that it can naturally account for changes in player performances over time without needing to model them explicitly.

Chapter 9

A Method for Estimating Individual Ratings from Group Competitions in Real-Time

Abstract

Providing direct and indirect contributions of more than $18 billion to the nation's gross output in 2004, the computer and video gaming industry is one of the fastest-growing sectors of entertainment. A large part of that market includes team-oriented online games. Players in these games often have a high-level of interest in statistics that help them assess their ability compared to other players. However few models exist that estimate individual player ratings from team competitions. The following presents an efficient recursive Newton-Raphson method for estimating player ratings in terms of how well the individual players contribute to their team's winning. In addition, parameters that estimate other characteristics of the games themselves are also analyzed. The model is posed in a Bayesian framework. The recursive Newton-Raphson method for estimating the model is efficient enough to be used real-time in order to improve gameplay in current matches and for helping players decide which matches to participate in. Companies and servers that apply well-developed statistics for assessing their players' abilities are more likely to attract and retain players, leading to greater success in the industry. As measured on 4,675 matches, the recursive Newton-Raphson method results in an accuracy of 70% when used to predict the outcomes of the matches used to estimate player ratings. In addition, it is shown

that this method is fast enough to be used in real-time whereas *Markov-Chain Monte Carlo* (MCMC) is not.

9.1 Introduction

According to one marketing research firm, the computer gaming industry is the fastest-growing sector of entertainment [18]. Robert Crandall, a senior fellow with the Brookings Institution, and Gregory Sidak, visiting professor of Law at Georgetown University stated in a recent study that "The entertainment software business provided direct and indirect contributions of more than $18 billion to the nation's gross output in 2004" [13]. In fact, "video games represent an $11 billion U.S. entertainment industry, larger than Hollywood box office sales" [18]. A large portion of this revenue comes from the online gaming sector, which is projected to bring in $2.5 billion in 2006 [38]. 44% of all video and computer gaming players say they play online in addition to offline games [71]. One of the more popular genres in online gaming involves team competitions. This includes games based on popular sports, but more popular are team-based first-person shooter games which include blockbuster titles like Halo 2 and Half-Life 2. Millions of "gamers" can be found playing these games every day.

Most online players prefer having a method of tracking their progress in the games they play, and comparing themselves to other players. If a player feels like he or she can achieve some goal with a given statistic, they are more likely to continue playing. Games and servers that provide better developed statistical methods for comparing a player's skill are more likely to retain a larger player base, and hence be more profitable. Therefore, having good ways of rating players benefits the players, the companies producing the games, and the companies running servers for the games. In addition, if aspects of the game besides the players can be judged statistically, the developers can use this information to improve the quality of newer games. For example, in this paper we show how the fairness of the "field" two teams play on can be analyzed. This can be used to improve the fairness of gameplay.

Most players of online team games play on "public servers". By public we mean that players are free to join, leave, and switch teams as often as they like. By server, we refer to the software that maintains the state for a given game and match and determines when a match is over and who the victors are. Every match is highly dynamic in terms of which players are on either team and therefore the ratings of the teams can not be estimated at the group level, but need to be built from individual ratings. Unfortunately, for team-based games, there are few systems in place that estimate individual player ratings from group competitions. The following presents an extension of the model proposed by [7] for paired-comparisons that estimates individual player ratings based on the winners of team competitions.

There are several types of statistics that can be tracked in these team competitive online games, many of which are interesting at the individual level, but are not consistent indicators of a player's team contribution. The goal of the model in this paper is to estimate how well a player contributes to his or her team winning the match. Therefore, the ratings are estimated only in terms of the wins a player achieves while playing on their given teams. It should be noted that the ratings here are not necessarily strong indicators of how these players would play one-on-one, but rather in team settings.

There are several reasons why ranking and rating players in public, online, team-based competitions can lead to an increase in the number of players that play and continue to play a given game or play on a given server:

- It gives players a measure of their performance and hence motivation to improve themselves. Our experience with running several online servers has shown that players tend to play more often on servers that provide a method of assessing their performance. The servers providing more statistical information are consistently full whereas the others are often near empty.

- It allows server administrators to judge the fairness of gameplay on their servers. Servers known for fair gameplay always attract more players than other servers. This is because the newer players know they have just as much a chance of winning on either team, and older players know the challenge level will remain consistent.
- It can aid players in choosing servers that better fit their abilities. If the easiness of a server can also be judged, then players can choose the servers that best fit their abilities. The following paper provides a model for comparing the difficulty level of a given server to other servers.

In the end, the games and servers that provide better statistical measures in addition to more challenging and fair gameplay will attract more players.

The game chosen for this paper is called Wolfenstein: Enemy Territory, also known as ET. It is one of the most popular team-based online games played with several hundred thousand unique players playing every day. Its popularity is partly due to the fact that it was made free by its publisher, Activision. In Enemy Territory, each match consists of an Axis side and an Allies side competing on a World War II historically inspired map. Each team has several objectives they need to complete within a certain time limit in order to win. Usually, one team's objective is to accomplish a given goal, while the other team's objective is to prevent them from accomplishing this goal during the time limit. This often unbalanced play-style, combined with the common coming and going of players on a public server, creates matches that require rich models in order to predict. They can be compared to a soccer game where either team can have as many players as they like so long as the combined number of players on the field is less than some maximum. To complicate matters further, the players are free to switch between teams at will. Furthermore, the soccer field does not have to be symmetrical: the field can be on an incline such that the ball naturally rolls towards one team's goal, and it can also include obstacles

on one end that are not on the other. The pool of possible players also does not have a limit, ranging as high as 1,000, and the players are free to leave and join a match at any time. There are no other published models designed to handle this complex of a competition. The following presents a model to estimate individual ratings from team-play and account for both the imbalance in the play-style, and the dynamic nature of the teams.

With the model presented here, we are able to answer several questions, including, "Who are the best players on this server?", "Which maps are the most difficult for the Axis side?", and "Which server is easier to play on for the average player?", in addition to questions like "How likely are the Allies to win the current match on this server?"

In a previous paper [53], we fit the model from this paper using MCMC. However, in order to give real-time statistics to players, we need a method that can estimate the ratings of up to millions of players on thousands of servers. MCMC would take longer than the length of a single match in most online games, and therefore could never give up-to-date ratings. This paper presents a recursive Newton-Raphson method for quickly estimating the ratings of large amounts of players across large amounts of servers. In order to be practical, any model for rankings individual players from team competitions needs to be efficient in both its memory requirements and speed. We show that the recursive Newton-Raphson method we propose is fast enough to fit the data in real-time, whereas MCMC is not.

This paper proceeds as follows. Section 9.1.1 gives related work, section 9.2 describes the data analyzed, the model is presented in section 9.3, section 9.4 explains the how the model parameters are estimated efficiently, section 9.5 shows the results of the estimation, potential applications and uses for an efficient estimation method for rating players is shown in section 9.6, and section 9.7 gives conclusions and directions for future work.

9.1.1 Related Work

Statistically motivated rating systems are not new. One of the earliest examples is that of [22] for chess ratings. He used the equivalent of a *Thurstone Case V* model which assumed the chess competitors had a rating score that followed a normal distribution. This model also assumed a normal distribution on the difference in player ratings. More recent versions of Elo's chess rating system have adopted a logistic distribution instead of a normal distribution because it has been shown that "weaker players have significantly greater winning chances" [73]. The logistic distribution version of Elo's model is equivalent to the model proposed by [7] for paired-comparisons.

The basic model used in this paper is an extension of the commonly used Bradley-Terry model. In this model, two opponents have ability parameters λ_1 and λ_2, and the probability of the first opponent winning is $\lambda_1/(\lambda_1 + \lambda_2)$. Several reviews on uses of and extensions to Bradley-Terry models can be found in the literature [7, 17, 16, 37]. One common extension proposed by [1] is to add a parameter to the model for home field advantage. The following uses a similar approach, but instead of a single parameter, there are two per map—one for the Axis side and one for the Allies side. In addition, [28, 26] recently extended the Bradley-Terry chess rating model to take into account uncertainty about a player's rating based on the amount of time that has passed since a player has competed.

What makes this work unique is that it presents a new model for estimating individual ratings from team competitions. Few other models have been presented for this situation. One notable model is that of [36]. They use Bradley and Terry's original formulation of the model, but obtain each team's λ by summing the ratings of its players: $\lambda_1 = \sum_{i \in 1} p_i$, where p_i is the rating of player i. This model fit well in their application which never had uneven amounts of individuals per group. It results in a linear increase in team rating when there are more players on one team than another. Experience has shown us, however, that having more players on one team does not

result in a linear, but rather exponential increase in the team's perceived ability to win for our application. Another similarity between our model and that of [36] is that we both present a method for weighting individuals by how much they are expected to contribute to their respective teams. [36] suggest the weighting be predetermined using expert knowledge about the data, whereas we use a simple exposure model that uses the percent time a player plays on a given team for a given match as that player's "weight" for that match.

Another recently proposed model that estimates individual ratings from team competitions is that of [33]. Their ratings system is similar to the Thurstone Case V model given by [22] except instead of comparing two individuals, it compares two groups by summing the ratings of the individuals. Since the likelihood they use is a normal, unlike the model given by [36], team ratings increase exponentially when one team has more players than another. The main difference in the model presented here is that we assume a logistic instead of normal distribution on the difference in team strengths. This results in our model being an extension of the Bradley-Terry model, whereas theirs is an extension of the Thurstone Case V model. We made this decision because we felt weaker teams were more likely to win than the normal distribution would predict.

The model used in this paper was originally presented in [53]. This model was fit using MCMC, but MCMC is impractical for online gaming because it takes longer to fit the model than it does to play a match. Therefore, the estimates after a new match would already be inaccurate by the time they were available. The following presents a method for estimating the parameters of the model in [53] that is efficient enough for real-time use. Other methods for quickly fitting Bradley-Terry models have been proposed by [37, 27, 28, 26]. The method here is similar to that proposed by Elo as examined in [27] and those suggested by [28, 26], however these methods do not naturally extend to estimate individual ratings from groups. In addition, these

methods either do not model the uncertainty in a player's rating [27], or they model it explicitly [28, 26]. To model the variance in our model's parameters, we instead apply a method based on the suggestion of [5] which is analogous to a recursive form of the Newton-Raphson method. It uses a running estimate of the Hessian matrix, whose negative inverse yields a multivariate gaussian approximation to the covariance matrix of the parameters. This method is detailed in section 9.4.

In addition to providing ratings for individual players from group competitions, the model proposed by [33] also provides an efficient method for fitting the ratings in real-time. Their method uses an efficient message passing algorithm viewing the model in terms of factor-graphs whereas the method presented here is based on a recursive formulation of the Newton-Raphson method. The method of [33] also models uncertainty explicitly whereas we derive it from an estimate of the Hessian in our Newton-Raphson method. Both methods find a multivariate normal approximation to the posterior mode of their respective models.

9.2 Data

We obtained data by parsing the log files from three Enemy Territory servers yielding around 100 matches per server. For each match, the log files list which server the match was played on, which map was played, which side won—Axis or Allies—the length of the match, which players participated, and how long each player spent on both teams. Here is an example of a log file entry from one match:

```
InitGame: ...\mapname\fueldump\...
Winner: AXIS Time: 1800000
Name: |R!P|Orpheo GUID DFBB5: Time Axis: 0 Time Allies: 1450200
Name: |R!P|Crazyeskimo GUID EF071: Time Axis: 1549800 Time Allies: 0
Name: sliveR GUID 0A589: Time Axis: 1614950 Time Allies: 0
```

```
Name: DaSaNi GUID 3F6C7: Time Axis: 1278400 Time Allies: 0
Name: <ETA>Ronik GUID DF58D: Time Axis: 1446150 Time Allies: 0
Name: pXo GUID 67F24: Time Axis: 0 Time Allies: 1491900
Name: -|S*S|-Chewie GUID 89E44: Time Axis: 1364050 Time Allies: 0
Name: [MIST]Lars GUID 54ECD: Time Axis: 1584700 Time Allies: 0
Name: SexY SpenceR GUID 828D1: Time Axis: 0 Time Allies: 1444250
Name: Gunny Highway GUID B560E: Time Axis: 0 Time Allies: 1541850
Name: <ETA>Gorm GUID C66B6: Time Axis: 0 Time Allies: 1470750
Name: livefastdieyoung GUID 0E0FA: Time Axis: 1527650 Time Allies: 0
Name: MAC 10 GUID AB76C: Time Axis: 0 Time Allies: 1482550
Name: Sexy hamodi GUID 90838: Time Axis: 510200 Time Allies: 0
Name: xbigphillx GUID 265DE: Time Axis: 0 Time Allies: 877700
Name:  Alone GUID F48ED: Time Axis: 0 Time Allies: 437900
Name: -=AoW=- Red GUID 566FC: Time Axis: 0 Time Allies: 0
Name: Boas7 GUID 1EF51: Time Axis: 0 Time Allies: 0
Name: Eryx GUID 77068: Time Axis: 0 Time Allies: 0
Name: Monoxide GUID 8D4BD: Time Axis: 0 Time Allies: 0
Name: ProfessorX GUID 70D81: Time Axis: 0 Time Allies: 169950
```

All but the map name information has been removed from the first line of the log file entry because only that information is currently used. The next line, or "Winner" line gives the side that won the match and how long the match lasted in milliseconds. In this match, the Axis side won and the match lasted 1,800,000 milliseconds, or 30 minutes. The remaining "Player" lines give information about each player. First, a name is given. These are nick names the players choose and often change. In order to track players despite changing names, the next field is used. A player's GUID or *Globally Unique IDentifier* is a 32-digit hexadecimal field that is unique across all servers for Enemy Territory and assigned by a central server for every player

everywhere in the world. The GUIDs presented here have been shortened to 5 digits for both privacy and brevity. The player records are stored based on this instead of their names which can change. After the GUID, the last two fields give the amount of time each player spent on the Axis or Allies team in milliseconds. For example, |R!P|Orpheo, the first given player, spent 1,450,200 milliseconds, or 24 of the matches' 30 minutes on the Allies side, which lost. |R!P|Crazyeskimo, the second player from the top, spent 1,549,800 milliseconds, or almost 26 of the matches' 30 minutes on the winning Axis side. Notice that neither player spent the entire 30 minutes of the match in the game. In addition, here is an example of a player in a different match who spent time on both sides:

```
Name: BlackSheep GUID 6C875: Time Axis: 552600 Time Allies: 1336200
```

Here, BlackSheep spends 9 minutes on the Axis team, and then 22 minutes on Allies team. It is not uncommon to see players that do not play the entire match on the same team and players who do not participate for the entire match in online matches like those in Enemy Territory. This happens for several reasons, including players joining the server late in a match, and players switching teams to play with closer acquaintances. Therefore, we present a method for handling this dynamic behavior in section 9.3.

9.3 Model

The following shows how we used the given data to estimate player ratings. We first present models for the individual player ratings, and then the likelihood for predicting whether a given team will win a match given the data above. We give three potential models for individual player ratings: a basic one in section 9.3.1, one that accounts for map-side effects in section 9.3.2, and a model that accounts for the difficulty of a given server in section 9.3.3.

9.3.1 Basic Model

Although the outcome of each competition is measured at the team- or side-level, the fundamental unit of the rating system needs to be built on the abilities of the individual players themselves. The likelihood given in section 9.3.4 determines a team's rating from the players. Therefore, we need a model for the individual player ratings. Initially, we let θ_i represent player i's ability to help their side win a match. There are no existing standards for rating players in these games and so for simplicity we chose a model that places player ratings symmetrically around 0. The better players will have positive player ratings, and the worse players negative ratings. This is in contrast to the ratings in, for example, chess, that have values ranging closer to 1500–2500. Our first model for player ability is:

$$\theta_i \sim N(\mu, \sigma_\theta^2). \tag{9.1}$$

σ_θ^2 is given a prior distribution and $\mu = 0$ without loss of generality. This model makes the naive assumption that a player's probability of winning will not be affected by the difficulty of the side that player plays on for a given map.

9.3.2 Accounting for Map-Side Effects

Because maps in Enemy Territory and most other games of this nature are so varied, it is convenient to describe the details of a given match in terms of the combination of map and side that each player played on. Throughout this paper, we will refer to a team or a player's "map-side" combination, meaning the map the match took place on, e.g. the map "venice", and the side the player played on—Axis or Allies. The model given above is naive considering most maps were designed such that one side has a major advantage. Since this imbalance is uniform by design for all players, a more robust model is to assume that each player's ability increases or decreases

according to a map-side rating. In this model, we let $\theta_{i,m-s}$ represent player i's ability to help side s win a match played on map m. As with player ratings, we choose a model that places map-side ratings symmetrically around 0:

$$\theta_{i,m-s} \equiv \theta_i + \gamma_{m-s} \text{ with } \theta_i \sim N(0, \sigma_\theta^2) \text{ and } \gamma_i \sim N(0, \sigma_\gamma^2). \tag{9.2}$$

This is similar to fitting the "homefield advantage parameter" proposed by [1], except that instead of one parameter per "field", here there are two—one for the Axis side, and one for the Allies. This is because it is not clear from the data which side has the advantage, and so we infer this by fitting a parameter for both sides instead of choosing a single side. We felt it easier to interpret each side's map rating separately, rather than have a rating for just one side. Although this does not model individual player-map-side interactions, it is assumed that the additive effect will be stronger than the individual interaction effects because most maps have been designed to be uniformly uneven. We are more interested in increasing the predictive power of the ratings than in the interactions themselves, and therefore we chose not to model them at this point.

With this model, a player's ability on a current map becomes his base rating, θ_i, offset by a map-side effect, $\gamma_{m,s}$. This information is interesting because it suggests that a team of skilled players can still enjoy a reasonable level of challenge against a team of less-skilled players as long as the map-side effects give the less skilled team an equivalent advantage. In addition, the estimates for $\gamma_{m,s}$ can be used to judge which maps are more balanced between Axis and Allies. Server administrators can use this information to choose which maps to place on their servers to improve gameplay, and map creators can use this information to better balance the gameplay in their maps.

9.3.3 Server Difficulty

If players can be ranked based on matches they participated in on a given server, it would be beneficial to also be able to compare players participating on different servers, or players that played on several servers. In order to determine how a given server affects a player's rating, the following model adds a server bias, ψ_j, into each player's rating, yielding:

$$\theta_{i,m-s,j} \equiv \theta_i + \gamma_{m-s} + \psi_j \text{ with } \theta_i \sim N(0, \sigma_\theta^2), \gamma_i \sim N(0, \sigma_\gamma^2) \text{ and } \psi_j \sim N(0, \sigma_\psi^2). \quad (9.3)$$

Note that the server difficulty is modeled as an increase instead of decrease a player's rating. This means that we are really modeling server "easiness." We did this for simplicity in the model. It just means that lower and not higher values of ψ denote a more difficult server. With this model, a player's rating depends on their base ability, the side and map they are playing on, and the server they are playing on. Besides being able to compare players across servers more effectively, estimating ψ also gives us a measure of how difficult each server is. This information can be used by newer players to choose easier servers, or more experienced players can use it to find challenging gameplay. In addition, since the player ratings can now be fit globally over all servers, a player can compare his or herself to all of the players in the game, instead of only those who play on a certain server. The accuracy of the comparison, however, will be affected by how often players move between servers. If enough server "cross-over" occurs, then the ratings should be reasonable enough for comparison.

Another interesting aspect of ψ is since the amount added to each side is equal to $N_s\psi$ where N is the number of players on side s, ψ can be interpreted as the rating increase a given side s expects for each additional player on that side. Servers where having additional players will not make up for the skill of the players are more difficult than those where a few extra players alone can decide the winner. Since

this additional information is interesting for rating a server alone, ψ is also fit in the single-server results given in section 9.5.

9.3.4 Likelihood

The likelihood we use is based on a modified version of the Bradley-Terry paired-comparison model [7]. In order to predict which of two teams is more likely to win, we choose a side s's probability of winning a match played on map m to be proportional to $\lambda_{s,m} = exp(\sum_{i=1,i\in P_s}^{|P_s|} \theta_{i,m-s,j})$ where P_s is the set of players on side s and $|P_s|$ is the number of players on side s. Therefore the probability of side s_{Axis} defeating side s_{Allies} for the given map is $\lambda_{s_{Axis},m}/(\lambda_{s_{Axis},m} + \lambda_{s_{Allies},m})$. This results in the probability of a given side winning increasing exponentially with the size of that side compared to the other. This is appealing because unlike a game like chess where only one move can be made per side despite the number of actual players making the move decisions, in Enemy Territory and other real-time online team competitions, every player can act simultaneously. Having more players gives a team with more numbers the ability to more effectively defend or accomplish its objectives. It is rare for a team with two or less players than another to win unless that team is very good.

If G is the total number of matches, $w(g)$ gives the winning side for match g, $l(g)$ the losing side, and m the map played on, then this model gives rise to the following likelihood function:

$$P(w|\lambda) = \prod_{g=1}^{G} \lambda_{w(g),m}(\lambda_{w(g),m} + \lambda_{l(g),m})^{-1} \tag{9.4}$$

where

$$\lambda_{w(g),m} = exp(\sum_{i=1,i\in P_{w(g)}}^{|P_{w(g)}|} (\theta_{i,m-s,j})) \tag{9.5}$$

and

$$\lambda_{l(g),m} = exp(\sum_{i=1,i\in P_{l(g)}}^{|P_{l(g)}|}(\theta_{i,m-s,j})). \tag{9.6}$$

One problem with the above likelihood is that in public-style servers players can come, leave, and change teams almost at will. Therefore, a model that does not take into account how much time a player spends on each team is inaccurate. Since the match data we are using does indicate the amount of time the players spent per team, this information can be used to modify the λs above:

$$\lambda_{w(g),m} = exp(\sum_{i=1,i\in P_{w(g)}}^{|P_{w(g)}|}(\tau_{i,w(g)}\theta_{i,m-s,j})) \tag{9.7}$$

$$\lambda_{l(g),m} = exp(\sum_{i=1,i\in P_{l(g)}}^{|P_{l(g)}|}(\tau_{i,l(g)}\theta_{i,m-s,j})) \tag{9.8}$$

where $\tau_{i,w(g)}$ ($\tau_{i,l(g)}$) is the percent of the total match time player i spent on the winning (losing) team, $w(g)$ ($l(g)$). This yields a simple approximation to integrating the player abilities over the time of the match to estimate the overall strengths λ per team.

9.3.5 Prior selection

Since the prior means for the parameters are all chosen to be 0 without loss of generality, the remaining priors to choose are on the variances of the player ratings, σ_θ^2, the map-side ratings σ_γ^2, and the server ratings σ_ψ^2. In [53], instead of choosing non-informative priors for these variances, we placed hyperprior distributions on them using inverse gamma distributions, with α and β chosen such that the distributions have means of 1.0 and variances at 1/3. This was done simply to keep most player ratings between -3 and 3. Again, we chose priors this way because there are no current standards for rating players in these types of competitions, and so we decided

for simplicity to assume they follow close to a standard normal distribution. We used hyperpriors so we could observe the relative differences in the standard deviations.

Here, we set the priors on the variances slightly larger than the ones obtained using MCMC, since the data sets used with MCMC estimation were much smaller. We did this instead of using hyperpriors because 1) we had prior knowledge from the previous work with different data in [53], and 2) because it simplifies our updating method. A future version will incorporate hierarchical terms into the estimation method.

9.4 Efficiently Estimating the Parameters

The method we use to estimate the posterior mode of the given model is a recursive version of the Newton-Raphson method. The standard Newton-Raphson method has the following form:

$$\theta_k = \theta_{k-1} - [L''(\theta_{k-1})]^{-1} L'(\theta_{k-1}) \tag{9.9}$$

where L' is the vector of the first derivatives of our model's log posterior with respect to the parameters and L'' is the matrix of second partial derivatives. Here, k refers to the current iteration of the method. It is worth noting at this point that $[-L'']^{-1}$ corresponds to the covariance matrix of a multivariate gaussian approximation to the posterior when evaluated at the mode. The following are two potential problems with using the full Newton-Raphson method with our application:

1. It would require storing information about every match ever played and then re-running the full Newton-Raphson method over all of the data after each match. This can quickly become impractical in terms of both time and memory requirements.

2. It requires storing the entire matrix of second partial derivatives. When dealing with millions of players, there will not be enough space to store it using current

technology, and not enough time to invert it before the next match begins—let alone re-run the fit after each match.

It would be preferable to have a method that did not require storing a history of the match and did also not require storing all of L''. One way around storing L'' is to use a diagonal approximation as suggested in [5]. This would make the assumption that there is little to no correlation between the parameters, and so the results may not be as accurate, but they may be reasonable.

To avoid storing a history of the matches, instead of using a full Newton-Raphson, we employ a recursive approach where the information from a single match is only used once in estimating the model parameters. The relevant parameters are updated after each match, and then that match is forgotten. The form for this, suggested by [5], is as follows:

$$\theta_t = \theta_{t-1} - \frac{1}{\tau}\phi_t L'_{t-1} \tag{9.10}$$

where

$$\phi_t = \left((1 - \frac{2}{\tau})\phi_{t-1}^{-1} + \frac{2}{\tau} diag(L''_{t-1})\right)^{-1} \tag{9.11}$$

and is a running "leaky" sum of diagonal approximations to $[L'']^{-1}$ taken from the last $\tau/2$ matches. Here t does not refer to the current iteration, but instead to the current match. Therefore, L' and L'' both refer to the first and second derivatives only with respect to the single match t. Dividing by τ instead of $\tau/2$ means the update will be half what the estimate suggests it should be. This is to offset the potential inaccuracy in estimating L'' in this manner. Although the updates are potentially less accurate, they do not require storing all of the elements of L''. The quantity $\phi_t/(\tau/2)$ is therefore a running and "leaky" average of a diagonal approximation to the covariance matrix. We found that for our application, setting $\tau = 50$ worked well, and therefore we averaged our variances over the last 25 matches. [5] showed both theoretically and

experimentally that this basic formulation asymptotically outperforms the standard Newton-Raphson method, or any iterative gradient-descent method.

The first derivatives and diagonals for the second derivatives after a single match are as follows where $I(S)$ is an indicator function that returns 1 is side S won the match in question, and 0 otherwise:

First Derivatives:

$$\frac{dL}{d\theta_i} = \tau_{w,i} - \frac{\tau_{w,i}\lambda_w + \tau_{l,i}\lambda_l}{\lambda_w + \lambda_l} - \frac{\theta_i}{\sigma_\theta^2} \tag{9.12}$$

$$\frac{dL}{d\gamma_{m-s}} = I(S) * |S|_s - \frac{|S|_s\lambda_s}{\lambda_w + \lambda_l} - \frac{\gamma_{m-s}}{\sigma_\gamma^2} \tag{9.13}$$

$$\frac{dL}{d\psi_j} = |W| - \frac{|W| * \lambda_w + |L| * \lambda_l}{\lambda_w + \lambda_l} - \frac{\psi_j}{\sigma_\psi^2} \tag{9.14}$$

Second Derivatives (diagonals only):

$$\frac{d^2L}{d\theta_i\theta_i} = -\frac{(\lambda_w + \lambda_l)(\tau_{w,i}^2\lambda_w + \tau_{l,i}^2\lambda_l) + (\tau_{w,i}\lambda_w + \tau_{l,i}\lambda_l)^2}{(\lambda_w + \lambda_l)^2} - \frac{1}{\sigma_\theta^2} \tag{9.15}$$

$$\frac{d^2L}{d\gamma_{m,s}\gamma_{m,s}} = -\frac{(\lambda_w + \lambda_l)|S|^2\lambda_s + (|S|\lambda_s)^2}{(\lambda_w + \lambda_l)^2} - \frac{1}{\sigma_\gamma^2} \tag{9.16}$$

$$\frac{d^2L}{d\psi_j\psi_j} = -\frac{(\lambda_w + \lambda_l)(|W|^2\lambda_w + |L|^2\lambda_l) + (|W|\lambda_w + |L|\lambda_l)^2}{(\lambda_w + \lambda_l)^2} - \frac{1}{\sigma_\psi^2} \tag{9.17}$$

$$\tag{9.18}$$

One problem with applying this approximation to a Bayesian framework is that there are "shrinkage" terms in the derivatives that in a standard Newton-Raphson method are added once per iteration given an entire set of matches. These are the last terms in each derivative equation. In our recursive Newton-Raphson method, there is only one iteration, but that iteration can be infinitely long. Therefore, the shrinkage terms should only be added once over the course of training. In the second derivatives, the shrinkage terms are constant and can be correctly added at the beginning of our method before any updates. In the first derivatives, however, the

shrinkage is proportional to the current value of the parameter, and therefore can not be added beforehand. One way to approximate adding in the variable shrinkage terms is to divide the current update's shrinkage amount by the total number of updates that parameter will receive. However, the number of updates given this recursive formulation is potentially infinite, and certainly not known a priori. Therefore, we choose instead to weight the shrinkage term in each update by the infinite geometric series 2^{-t-1} which converges to one. This has the effect of adding the shrinkage term to the parameter only once over the course of estimating that parameter. It is also an appealing formulation in the Bayesian sense because the effects of the shrinkage terms will become less pronounced as more real data is observed.

One benefit of using a recursive algorithm like this is that it naturally tracks time-varying differences in both player, map, and server ratings, and also tracks changes in the standard deviations of each. If these quantities change over time, the update will create a new rating that is a mix of the old estimate and the current estimate. The variance estimates are averages over the last 25 matches, and so if a player begins playing far differently than he or she played before, their variance can increase and allow their rating to change more rapidly to track this.

Another benefit of this approach is its efficiency in terms of memory and computational time. Because we use a diagonal approximation to the covariance matrix, the only quantities stored are a rating and variance for each parameter. The updates themselves take time proportional to the number of players in a single match, a value that never exceeds 32 in Enemy Territory. Other games and servers also have upper limits on the number of players in a match. Therefore, the update is done in constant time.

The resulting recursive algorithm yields a method for estimating player, map, and server ratings efficiently over a potentially infinite data set. Experimental results of using this method are given in the next section.

9.5 Results

In this section we first discuss the complexity of the recursive Newton-Raphson method compared to MCMC in section 9.5.1. We then derive rankings for players based on match information on three different Enemy Territory servers. First, rankings are derived for each server separately in section 9.5.2, and then we rank the players across all three servers simultaneously in section 9.5.3. The map-side effects are analyzed in section 9.5.4 and section 9.5.5 reports on how MCMC and the recursive Newton-Raphson method compare in terms of accuracy.

9.5.1 Complexity Comparison

The main reason for choosing a recursive Newton-Raphson method in section 9.4 instead of the MCMC method used by [53] is that MCMC takes longer to fit the data than the length of the average match in Enemy Territory, which is 15 minutes. Here we give a brief overview of the complexity of both MCMC and the recursive Newton-Raphson instead of an in-depth analysis because the difference is so pronounced.

The complexity of MCMC for this application is proportional to NPM where N is the number of iterations chosen for the chains, P is the number of players who play over all the matches, and M is the average number of matches per player. The small data set used by [53] consisted of 300 matches, 877 players, and 7 matches on average per player. Fitting this data using 100,000 iterations of MCMC took well over 15 minutes on a 2.0 Ghz Opteron. This is longer than the average length of a match in Enemy Territory. The data used in this paper consists of nearly 5,000 matches and over 2,000 players, and could therefore take nearly 2 orders of magnitude longer to fit than the previous data set. The length of time it takes to fit the model using MCMC is too long to be used in this real-time application.

The complexity of the recursive Newton-Raphson method we give in section 9.4 is proportional to the number of players in a given match—in Enemy Territory

this never exceeds 64—therefore the update time is at worst a small constant and takes less than 1 second to process on a 2.0 Ghz Opteron. Since the complexity of the recursive Newton-Raphson method scales only linearly with the size of the number of competitors, it can be extended for use in applications that are significantly larger than this one without a major increase in running time. This new method can be applied real-time because it can update the model before the next match completes. The comparison may not seem completely fair because we are comparing the update from a single match to fitting the entire data set as is done in MCMC. However, using MCMC would *require* refitting on the entire data set after every match. That said, the amount of time it takes to fit the larger data set used in this paper with the recursive Newton-Raphson method is 1.5 minutes on the same 2.0 Ghz Opteron. This is compared to an estimate at nearly a day using MCMC. Therefore, this method is significantly faster than MCMC and is fast enough to be used in real-time.

9.5.2 Separate Server Rankings

In this section we present the top ten rankings of the players for each server. The model was fit once per server using matches from only that server. The approximation method given in the previous section was used to update the participating parameters after each match. By participating parameters we mean those corresponding to the players, map-sides, and server for that match. The results are shown in tables 9.1–9.3. The players are conservatively ranked by how they play two standard deviations below their posterior modes. This has the desired effect of penalizing players whose ratings are less certain. It is not uncommon in these tables to see players who have worse winning percentages ranked higher than those with better ones. This suggests that their wins were against more difficult odds—e.g. harder map-side combinations, better players, etc.

Table 9.1: Ratings and rankings for server 1. Shown are the player ranks, the posterior means of their base ratings, θ_i, the standard deviation of each player's θ_i, the nicknames they use in-game, the number of games each played, and their winning percentage. The same information is shown in tables 9.2–9.3 and later in 9.4

Rank	Post. Mean	SD	Name	Games	Winning %
1	1.41	0.78	|R!P|Burnz	55	0.72
2	1.34	0.77	[ccp]Bob-Brinks	83	0.68
3	1.07	0.69	CoRnEdBeEf&CabBagE	46	0.73
4	1.19	0.84	Papoose	20	0.72
5	0.89	0.7	-|S*S|-dgifTy!	44	0.59
6	0.96	0.77	-|S*S|-Bari	33	0.57
7	0.86	0.74	SexY SpenceR	24	0.66
8	0.98	0.87	Ignis	6	1.00
9	0.67	0.74	=LK3=GibS	34	0.66
10	0.94	0.9	BlackSheep	7	0.90

Since the players who play commonly on public servers rarely receive popular attention, the aliases used will not likely be recognizable to the general public. However, the goals of giving each player a rating and hence a stronger incentive to continue playing and improving his or her abilities in the game, in addition to better predicting the outcome of the games, have been met. That said, the players shown are known to be highly skilled amongst those who play on the servers.

9.5.3 Combining all 3 Servers

To determine which were the best players over all three servers, we took advantage of the fact that many players play on more than a single server and ran our method over all three servers simultaneously. The results are shown in tables 9.4–9.5.

The server results are surprising at first because most players consider the settings on server 1 to make it a more difficult server. However, the fact that the matches on server 1 are generally smaller because it is the least popular of the servers may contribute to it being easier overall for those players who play on multiple servers. It is not as surprising to see server 2 listed as an easier server since it is designed to

Table 9.2: Ratings and rankings for server 2.

Rank	Post. Mean	SD	Name	Games	Winning %
1	2.37	0.73	Carnaugh	226	0.55
2	2.44	0.77	[eB]=Eric=	120	0.62
3	2.65	0.91	[ELC]Crazyeskimo	173	0.60
4	2.34	0.8	FragFumbler	112	0.62
5	2.03	0.68	mario	129	0.69
6	1.94	0.74	Saruman	127	0.59
7	2.14	0.88	freedom4me	239	0.60
8	1.83	0.75	<ETK>Soldier!Ryan	185	0.63
9	2.11	0.9	-|S*S|-WizKid	163	0.61
10	2.02	0.92	Hosj	67	0.69

Table 9.3: Ratings and rankings for server 3.

Rank	Post. Mean	SD	Name	Games	Winning %
1	2.16	0.81	Sexy Bro Two	31	0.82
2	1.87	0.85	Samson	141	0.56
3	1.56	0.75	Private Jackson	181	0.58
4	1.51	0.78	lauchaxXx	56	0.64
5	1.44	0.76	b!t	72	0.71
6	1.64	0.88	K4F*Slayer	70	0.64
7	1.61	0.88	PBRStreetGang CDN	125	0.60
8	1.38	0.78	[BoD]hamodi	136	0.55
9	1.27	0.73	C-Hu$tle	33	0.70
10	1.53	0.87	K4F*Boas7	63	0.77

Table 9.4: Ratings and rankings for all three servers fit simultaneously.

Rank	Post. Mean	SD	Name	Games	Winning %
1	2.81	0.82	[eB]=Eric=	154	0.61
2	2.42	0.73	Carnaugh	226	0.55
3	2.45	0.75	FragFumbler	112	0.62
4	2.59	0.91	[ELC]Crazyeskimo	277	0.57
5	2.02	0.72	mario	161	0.66
6	2.2	0.81	Sexy Bro Two	31	0.82
7	2.02	0.73	Saruman	127	0.59
8	2.02	0.76	<ETK>Soldier!Ryan	195	0.64
9	2.16	0.9	freedom4me	242	0.60
10	1.94	0.89	lauchaxXx	155	0.61

be an "easier" server and therefore attracts more players that are new to the game. Server 3 is known to have some of the better players from all the servers and also has an "eccentric" administrator. Since it attracts primarily more experienced and server-loyal players instead of newer ones, ranking the third server as the hardest is reasonable.

Here we see an example where players could use this information to choose a server. The better players could choose to play on the 3rd server to enjoy more of a challenge, whereas newer players may be likely to choose the second server because it is easier and it also usually has more players on it. The method we use to estimate these parameters is efficient enough to keep up with millions of players and thousands of servers, and therefore it can be applied to any one of the popular games that are already out or scheduled for release in the near future.

9.5.4 Map-Side Effects

One of the interesting benefits of the model used is that rating parameters are also fit for each map-side combination. This information can be used to judge which maps are more even and which maps have the least balance between Axis and Allies. Tables 9.6–9.9 show the results of fitting the map-side parameters on each server for first the

Table 9.5: This table shows the servers ranked by difficulty, giving the most difficult first. Also shown for each server is the posterior mode, the standard deviation, and which server is in question.

Rank	Post. Mode	SD	Server
1	1.46	0.67	3
2	2.35	0.79	2
3	2.45	0.89	1

Table 9.6: Map-side ratings for first the most uneven map and then the most even map on server 1. The rows show the in-game name of the map, the side of the map described by the current row, the posterior mode of the map's rating or γ for that side, and the standard deviation of that map's rating. Tables 9.7–9.9 show the same information for each's respective data.

Name	Side	Post. Mode	SD
fueldump	Allies	0.09	0.13
fueldump	Axis	-0.09	0.13
venice	Allies	-0.0	0.13
venice	Axis	-0.03	0.13

most uneven map on that server, and then the most even map. In both cases, maps that are not common to all three servers were not considered. In general, the results are in line with the perception of the players. Fueldump is a long map, but it can be won at almost any time. The tc_base map is usually easier for the Allies to win, however it had been modified prior to these matches to be more even, and the results suggest this modfication has been successful. The same goes for the radar map. The venice map is known to be one of the most even and enjoyable maps made, so it's not surprising to find it listed as is.

From these findings, it is not unreasonable to conclude that, for example, fueldump is imbalanced in favor of the Axis and that tc_base is probably a fair map. Server administrators could decide to either not include maps like fueldump in the list of maps on their server, or they can take measures to ensure that when it is played,

Table 9.7: Map-side ratings for first the most uneven map and then the most even map on server 2.

Name	Side	Post. Mode	SD
fueldump	Allies	0.11	0.06
fueldump	Axis	-0.1	0.06
radar	Allies	0.01	0.11
radar	Axis	-0.02	0.11

Table 9.8: Map-side ratings for first the most uneven map and then the most even map on server 3.

Name	Side	Post. Mode	SD
fueldump	Allies	0.16	0.1
fueldump	Axis	-0.13	0.1
tc_base	Allies	-0.02	0.1
tc_base	Axis	0.01	0.11

Table 9.9: Map-side ratings for first the most uneven map and then the most even map on server 1. The rows show the in-game name of the map, the side of the map described by the current row, the posterior mode of the map's rating or γ for that side, and the standard deviation of that map's rating. Tables 9.7–9.9 show the same information for each's respective data.

Name	Side	Post. Mode	SD
fueldump	Allies	0.12	0.06
fueldump	Axis	-0.09	0.06
tc_base	Allies	-0.02	0.08
tc_base	Axis	0.04	0.08

the Axis team consists of more skilled players. Map makers can use this information to consider whether they should make changes to fueldump that would make it more even given equally skilled teams. Games with statistically driven level design are more likely to attract players, as are servers that include more balanced maps in their lineups.

In addition, because the method we use to estimate the map ratings is efficient enough to run real-time, server administrators can use the map rating information to judge the fairness of a current match in progress, and make changes to improve balance. More on this is discussed in section 9.6.

9.5.5 Measuring Performance

The performance of the models was measured by their accuracy in predicting the matches used to estimate the model parameters. In order to remain unbiased, the prediction measurement for a given match is taken *before* updating the model based on that match. The results are shown in table 9.10. Notice that the accuracy in predicting matches on server 1 is higher than the others. This may be due to the fact that the matches on that server consist of less players, and there are less players overall on that server. This lowers the amount of variability in predicting a given match, and therefore leads to a slightly higher accuracy. Given the complexity of these diverse online matches, an unbiased, first-pass accuracy of near 70% is significant. In addition, 70% is comparable to the results obtained by [33] on an application with less complexity than the one used here.

It is important to note that the accuracy reported by [53] was closer to 100% when using MCMC. However, the MCMC fit had access to all of the data simultaneously, and therefore it could predict a match's outcome *after* updating the model based on that match. This results in an unrealistic positive bias when using MCMC. Fitting using MCMC could be tested using the same methods as the updating method

Table 9.10: This table shows posterior mode prediction accuracy for each server. The accuracy for a given match prediction was measured *before* updating the model parameters based on that match.

Server	Accuracy
1	0.72
2	0.70
3	0.69
All	0.70

here, however the amount of time to perform such a comparison makes it impractical. In addition, since MCMC can not be used in a real-time situation, its accuracy is, in a sense, not as important for this application.

Another, more adhoc performance measure is to simply verify that the values of the parameters and resulting rankings follow the intuitive sense of those familiar with the servers, maps, and players. Acting as those who have experience with these maps, servers, and players, we agree that, in general, the rankings fit.

9.6 Applications

This section describes potential applications for the suggested model and parameter estimation method.

9.6.1 Ranking the Players

Our first application is the obvious one, using the ratings to rank the players on the servers and across the servers. Players tend to prefer servers that give them a ranking they can work to improve. The proposed method is efficient enough to provide real-time rankings and ratings to players on single servers, as well as across multiple ones. These ratings and rankings can be updated quickly after every match, and made available to the players for comparisons. For example, for the servers that

we run, players can go to web pages to see listings of their ratings and rankings and compare them to other players on the servers. In addition, we provide methods for players to view their ratings and rankings with in-game commands. The players tend to appreciate these features. For example, server 2 is almost consistently full at 25-32 players, whereas server 1, where we do not run an in-game command for viewing their rankings, is almost always empty. Having the ability to query a player's rating information in-game is probably not the only factor contributing to server 1's lower popularity, but it is worth noting.

Because we model server difficulty as well as player and map-side ratings, players can compare themselves across servers when they are fit simultaneously. This assumes there are enough players that play on more than one server to make comparisons meaningful. For example, if *none* of the players who play on the third server ever play on any other server, their ratings and their server's difficulty rating can not be correctly compared to other players and servers.

9.6.2 Choosing Servers

Since we fit a server "easiness" parameter with ψ per server, we can use that parameter to rank the servers in order of difficulty. The parameter estimation method we used is efficient enough to update ψ for a large amount of servers in real-time after each match. Games that have this information available real-time can make it available to players who are trying to choose which servers to play on. Newer players can choose the easier servers, and veterans can choose servers that will give them more of a challenge. Also, because player ratings are fit taking into account the difficulty of the servers they play on, the servers can be also be listed in order of the average player ratings of the players *currently* playing. This measure gives an even better estimate of the current state of the server because it represents skill of the players in the current match, instead of the average difficulty. Players can sort the servers by

the average rating of the players on them, and then choose servers with ratings to suit the desired challenge level. As a more advanced option, games can be designed to automatically choose servers that best fit a given player. This can ensure that players always have a positive experience in playing these complex team-based games.

Besides a more consistent and balanced gameplay experience, giving players the ability to choose servers based on difficulty gives them another incentive to continue playing. Players will start on easier servers with the goal of improving their skills to a point they can comfortably play on harder servers. This natural improvement path will encourage players to return again and again to a game to see if they are good enough to play on harder and harder servers. One of the most popular online game types is the MMORPG or Massively Multiplayer Online Role-Playing games. The most popular of these games, World of Warcraft, boasts a player base of more than 6 million [75]. The draw to the MMORPG is that the game is designed with explicit character development paths that give players incentives to continue playing. However, these paths can be very time-consuming to pursue, especially for the more casual gamers that play in games like Enemy Territory. The server rating system can supply a character development path for the casual gamer based solely on the ratings system. This can enlarge the audience for first-person shooters and encourage players to buy games with this feature, and continue playing them.

9.6.3 Balancing Teams

At the server level, administrators can use real-time player rating information to balance the teams currently playing. To make this easier, the game can use the likelihood model with the estimated posterior mode to give posterior prediction estimates on which team is likely to win. If a team is very unlikely to win the match based on this real-time information, administrators can move better players to the disadvantaged team, and / or move worse players to the favored team. This can also be done au-

tomatically. If a team falls below a specified probability of winning, say 30%, then the game can automatically move players to bring the probability as close to 50% as possible. A computer can easily try all possible moves involving one player and in such a way greedily optimize the probabilities around 50%. Servers that employ automatic team balancing will enjoy a consistently balanced level of gameplay and attract more players. Games that have this option available may be more popular than those that do not.

In addition to making the game more enjoyable for its players, balancing teams in this manner can improve the efficiency of rating the players. As [33] observed, "the [team balancing] process can be viewed as a process of sequential experimental design" [24]. This is appealing and "since the quality of a match is determined by the unpredictability of its outcome, the goals of [balancing teams] and finding the most informative matches are aligned!" [33]. If the data used to fit the model comes from matches where a form of team balancing has been employed, less data may be needed to achieve a more accurate fit of the model.

One important question when choosing a limit for automatic team balancing is whether or not the predictions are precise around that threshold. It is important that teams with, for example, a 70% chance of winning actually do win 70% of the time. We can report that, in our experiments, we found that when the prediction for a team winning was above 70%, that team did in fact win 70% of the time. Therefore, 70% seems to be a good value as a cut-off if more than a 70% winning chance is considered unfair.

9.7 Conclusion and Future Work

We have presented an efficient method for estimating the parameters of a model that gives individual ratings in team competitions. These ratings allow players to effectively track their ability to help the teams they play on win. The models presented

are also able to account for both the dynamic nature of the teams in public, online team-competitions, and the imbalance commonly associated with the levels or maps designed for these competitions. In addition, we have estimated parameters that allow us to analyze the fairness of the maps in terms of the sides playing. Since the estimation method is efficient, it can be used to provide real-time information about individual player rankings, server difficulty, and the fairness of the current match on a given server. The MCMC method used in the previous paper is not efficient enough to do this real-time. We have given applications that can use this information, including suggesting games allow players to choose which servers to play on based on server difficulty or the average rating of the players currently playing on a given server. We have also suggested that the information be used to automatically balance current matches by moving players between teams. Games and servers that provide well-developed methods for both tracking their players' abilities and ensuring balanced gameplay are more likely to attract players and therefore be more profitable.

Future work will first evaluate the models that the estimation method is used to fit. For instance, instead of depending upon the recursive algorithm itself to track time-varying changes in player abilities, these could be modeled explicitly. [26] suggested a model to do this for rating and ranking chess players.

It would also be interesting to model the effect of the number of players on how hard a given map-side combination is. There may be maps that are easier for a given side when there are fewer players, but become harder as the total number of players increases. In this model, γ_{m-s} would be the result of a regression fit instead of a single, learned constant. Being able to analyze the effects of the number of players on a maps fairness will allow server administrators to choose maps more appropriate for the number of players commonly on their servers, again leading to increases in server usage.

There is also an area where the estimation method could be improved. The one given in this paper was not designed to handle models posed in a hierarchical framework. This is because, like the "shrinkage" terms in the derivatives for this method, the updates for the hyperparameters should only be applied once over the course of the entire estimation. Therefore, we will derive a method for applying a single update equivalently over the course of training. This method will most likely be similar to the converging geometric series used for the shrinkage terms. We have already made a few attempts at doing this, but the results have not been stable enough to report at this point.

Part IV

Conclusion and Future Work

Chapter 10

Conclusion and Future Work

10.1 Conclusion

This dissertation has presented oracle learning. It has shown how oracle learning can be successfully applied to reducing the size of artificial neural networks, approximating domain experts, and adapting a machine learning algorithm to a given problem. Oracle learning can be an important tool for improving the results of standard machine learning.

10.2 Future Work

Future work in oracle learning will begin where SOL-CTR leaves off. This means more methods will be developed for adapting the problem to the learner. This will initially result in 1) different statistical methods for judging an artificial neural network's confidence in its outputs and 2) methods that adapt targets to the learner that do not use a confidence-based measure. One way to learn targets without using a confidence measure is to iteratively refine them throughout the training process. Training would update both the weights and the targets simultaneously, attempting to minimize classification error as the objective.

In addition to improving artificial neural network training, methods will be developed that can adapt other machine learning algorithms to given problems. One research direction that will be followed is developing data-driven customized distance

metrics for instance-based learning algorithms. It is well known that customized distance metrics can improve the performance of instance-based learning methods, but the customization process is time-consuming and often requires expert knowledge about the data. Automating this process will result in increased performance over non-customized metrics, and save time over manual customization. It will be determined whether oracle learning will continue to play a role in these methods, or if a new approach needs to be developed. Having ways of adapting problems to their learners results in machine learning algorithms that can achieve improved generalization accuracy over a wider range of problems.

Future work with paired-comparisons will take two initial directions. The first direction deals with including additional important aspects of gameplay that are not currently considered. For example, the difficulty for a side to win on a given map is often proportional the total number of players currently competing. If the number of players that play in a match for a given game varies greatly over the course of a day, then the effect of the total number of players on gameplay should be considered. Another aspect of gameplay that is not yet accounted for is interactions among the players that result in non-linear contributions of their ratings. Modeling these additional aspects will not only lead to improved gameplay for the players, but more principled approaches to understanding how various aspects of these complex games can affect play in general.

Another area that will be researched is to develop formal methods for analyzing the optimal way to use match prediction information in real-time for balancing teams. Instead of using the greedy ad hoc method proposed in chapters 7 and 9, the effects of moving players will be analyzed from a control theory or optimal experimental design point of view. First, methods for judging the effectiveness of the balancing will be developed. One way to do this in Enemy Territory is to observe how often each side wins on a given map with and without automatically balancing the teams. If the

balancing system is effective, it will result in the frequency of both teams winning being near 50%. Once methods for judging the performance of the team balancing or team control system are developed, formal methods will be applied to use this feedback to dynamically improve it. Applying these more principled methods to analyzing the balancing system can result in a more dynamic and improved gameplay experience for the players, and therefore lead to better team-based competitive online games in general.

Since the methods presented worked as well as they did in rating and ranking players in the complex application of team-based online gaming, they will potentially perform well on other complex applications of rating individuals in group competitions. Data will be collected in both the sports and work group arenas to determine if athletes and likewise employees can be rated in a more formal manner than they are currently. This can result in "dream teams" in sports chosen statistically, or more productive teams of employees in the work environment. These models will become especially powerful for these applications as effective methods for modeling how individual interactions contribute to team strength are also developed. Modeling these interactions will allow groups to be chosen not only based on how well individuals perform alone, but on how well they work with specific colleagues. It can be argued that rating individuals based on how well they work in a group is more important in the real-world than judging people on how well they work alone.

References

[1] A. Agresti. *Categorical Data Analysis*. Oxford University Press, 1988.

[2] Peter L. Bartlett. For valid generalization the size of the weights is more important than the size of the network. In Michael C. Mozer, Michael I. Jordan, and Thomas Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, page 134. The MIT Press, 1997.

[3] Kristin Bennett and Ayhan Demiriz. Exploiting unlabeled data in ensemble methods, 2002.

[4] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *COLT: Proceedings of the Workshop on Computational Learning Theory, Morgan Kaufmann Publishers*, pages 92–100, 1998.

[5] Leon Bottou and Yann LeCun. Large-scale on-line learning. In *Advances in Neural Information Processing Systems 15*. MIT Press, 2004.

[6] A.P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *PR*, 30(7):1145–1159, July 1997.

[7] R. Bradley and M. Terry. The rank analysis of incomplete block designs I: The method of paired comparisions. *Biometrika*, 39:324–345, 1952.

[8] L. Breiman. Bagging predictors. *Machine Learning.*, 24(2):123–140, 1996.

[9] Rich Caruana. Algorithms and applications for multitask learning. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 87–95, Bari, Italy, 1996.

[10] Rich Caruana. *Multitask Learning.* PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburg, PA, 1997.

[11] Rich Caruana, Shumeet Baluja, and Tom Mitchell. Using the future to "sort out" the present: Rankprop and multitask learning for medical risk evaluation. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in*

Neural Information Processing Systems, volume 8, pages 959–965, Cambridge, MA, 1996. The MIT Press.

[12] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 161–168, New York, NY, USA, 2006. ACM Press.

[13] Robert W. Crandall and J. Gregory Sidakk. Video games: Serious business for America's economy, 2006.

[14] Mark Craven and Jude W. Shavlik. Learning symbolic rules using artificial neural networks. In Paul E. Utgoff, editor, *Proceedings of the Tenth International Conference on Machine Learning*, pages 73–80, San Mateo, CA, 1993. Morgan Kaufmann.

[15] Mark W. Craven and Jude W. Shavlik. Extracting tree-structured representations of trained networks. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 24–30, Cambridge, MA, 1996. The MIT Press.

[16] H. A. David. *The method of paired comparisons*. Oxford University Press, 1988.

[17] R. R. Davidson and P. H. Farquhar. A bibliography on the method of paired comparisons. *Biometrics*, 32:241–252, 1976.

[18] DFC Intelligence. Online game market, 2006.

[19] Thomas G. Dietterich. Approximate statistical test for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, 1998.

[20] P. Domingos. Knowledge acquisition from examples via multiple models. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 98–106, San Francisco, 1997. Morgan Kaufmann.

[21] D. Dorfman, K. Berbaum, and C. Metz. Receiver operator characteristics rating analysis: Generalization to the populations of readers and patients with the jackknife method. *Investigative Radiology*, 27:723–731, 1992.

[22] Arpad. E. Elo. *The rating of chess players: Past and Present*. Arco Publishing, New York, 1978.

[23] S. Fahlman. Faster learning variations on back-propagation: An empirical study. In D Touretzky, G. Hinton, and T. Sejnowski, editors, *Proceedings of the 1988 connectionist models summer school*, pages 38–51, San Mateo, 1989. Morgan Kaufmann.

[24] V.V. Fedorov. *Theory of Optimal Experiments*. Academic Press, New York, 1972.

[25] William Finnoff, Ferdinand Hergert, and Hans Georg Zimmerman. Improving model selection by nonconvergent methods. *Neural Networks*, 6:771–783, 1993.

[26] Mark Glickman. Dynamic paired comparison models with stochastic variances. *Journal of Applied Statistics*, 28(6):673, 2001.

[27] Mark E. Glickman. A comprehensive guide to chess ratings. *American Chess Journal*, 3:59–102, 1995. submitted.

[28] Mark E. Glickman. Parameter estimation in large dynamic paired comparison experiments. *Applied Statistics*, 48(3):377–394, 1999.

[29] Sally Goldman and Yan Zhou. Enhancing supervised learning with unlabeled data. In *Proc. 17th International Conf. on Machine Learning*, pages 327–334. Morgan Kaufmann, San Francisco, CA, 2000.

[30] T.L. Graves, C.S. Reese, and M. Fitzgerald. Hierarchical models for permutations: Analysis of auto racing results. *Journal of the American Statistical Association*, 98(462):282–291, 2003.

[31] J. B. II Hampshire and B. A. Perlmutter. Equivalence proofs for multilayer perceptron classifiers and the Bayesian discriminant function. In *Proceedings of the 1990 Connectionist Models Summer School, 1990. D. Touretzky, J. Elman, T. Sejnowski, and G. Hinton, eds. Morgan Kaufmann, San Mateo, CA.*, 1990.

[32] Trevor Hastie and Robert Tibshirani. Classification by pairwise coupling. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.

[33] Ralf Herbrich and Thore Graepel. TrueSkill^{TM}: A Bayesian skill rating system. Technical Report MSR-TR-2006-80, Microsoft Research, Microsoft Corporation, Redmond, WA, 2006.

[34] G. E. Hinton. Connectionist learning procedures. *Artif. Intell.*, 40(1-3):185–234, 1989.

[35] Tzu-Kuo Huang, Chih-Jen Lin, and Ruby C. Weng. Ranking individuals by group comparisons. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 425–432, New York, NY, USA, 2006. ACM Press.

[36] Tzu-Kuo Huang, Ruby C. Weng, and Chih-Jen Lin. Generalized Bradley-Terry models and multi-class probability estimates. *Journal of Machine Learning Research*, 7(Jan):85–115, 2006.

[37] D. R. Hunter. MM algorithms for generalized Bradley-Terry models. *The Annals of Statistics*, 32:386–408, 2004.

[38] Alex Jarret. IGDA online games white paper 2nd edition – march 2003, 2003.

[39] Valen E. Johnson. A Bayesian χ^2 test for goodness-of-fit. *The Annals of Statistics*, 32(6):2361–2384, 2004.

[40] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection, 1995.

[41] Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In John E. Moody, Steve J. Hanson, and Richard P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 950–957. Morgan Kaufmann Publishers, Inc., 1992.

[42] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 396–404, San Mateo, CA, 1990. Morgan Kaufman.

[43] Y. LeCun, J. Denker, S. Solla, R. E. Howard, and L. D. Jackel. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 598–605, San Mateo, CA, 1990. Morgan Kauffman.

[44] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In Genevieve B. Orr and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*, pages 9–50. Springer, 1996.

[45] G.R. Leonard and G. Doddington. Tidigits speech corpus, 1993.

[46] D. J. C. MacKay. Probable networks and plausible predictions — a review of practical Bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, 6:469–505, 1995.

[47] M.A. Maloof. On machine learning, ROC analysis, and statistical tests of significance. In *Proceedings of the Sixteenth International Conference on Pattern Recognition*, pages 204–207, Los Alamitos, CA, 2002. IEEE Press.

[48] Joshua Menke and Tony R. Martinez. Simplifying OCR neural networks through oracle learning. In *Proceedings of the 2003 International Workshop on Soft Computing Techniques in Instrumentation, Measurement, and Related Applications*. IEEE Press, 2003.

[49] Joshua Menke and Tony R. Martinez. Using permutations instead of student's t distribution for p-values in paired-difference algorithm comparisons. In *Proceedings of the 2004 IEEE Joint Conference on Neural Networks IJCNN'04*, 2004.

[50] Joshua Menke and Tony R. Martinez. Domain expert approximation through oracle learning. In *Proceedings of the 13th European Symposium on Artificial Neural Networks (ESANN 2005)*, 2005.

[51] Joshua Menke, Adam Peterson, Michael E. Rimer, and Tony R. Martinez. Neural network simplification through oracle learning. In *Proceedings of the IEEE International Joint Conference on Neural Networks IJCNN'02*, pages 2482–2497. IEEE Press, 2002.

[52] Joshua E. Menke and Tony R. Martinez. Artificial neural network reduction through oracle learning. *Neural Processing Letters*, 2006. Submitted.

[53] Joshua E. Menke, C. Shane Reese, and Tony R. Martinez. Hierarchical models for estimating individual ratings from group competitions. *Journal of the American Statistical Association*, 2006. . In preparation.

[54] R. Micheals and T. Boult. Efficient evaluation of classification and recognition systems, 2001.

[55] Kamal Nigam and Rayid Ghani. Analyzing the effectiveness and applicability of co-training. In *CIKM*, pages 86–93, 2000.

[56] Kamal Nigam, Andrew K. McCallum, Sebastian Thrun, and Tom M. Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 39(2/3):103–134, 2000.

[57] A. Van Ooyen and B. Nienhuis. Improving the convergence of the backpropagation algorithm. *Neural Networks*, 5:465–471, 1992.

[58] Justus H. Piater, Paul R. Cohen, Xiaoqin Zhang, and Michael Atighetchi. A randomized ANOVA procedure for comparing performance curves. In *Proc. 15th International Conf. on Machine Learning*, pages 430–438. Morgan Kaufmann, San Francisco, CA, 1998.

[59] Lutz Prechelt. Adaptive parameter pruning in neural networks. Technical Report TR-95-009, International Computer Science Institute, Berkeley, CA, 1995.

[60] Foster Provost, Tom Fawcett, and Ron Kohavi. The case against accuracy estimation for comparing induction algorithms. In *Proc. 15th International Conf. on Machine Learning*, pages 445–453. Morgan Kaufmann, San Francisco, CA, 1998.

[61] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[62] A.E. Raftery and S.M. Lewis. *Implementing MCMC*, pages 115–130. Chapman and Hall, 1996.

[63] R. Reed. Pruning algorithms—a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, September 1993.

[64] Y. Reich and S. Barai. Evaluating machine learning models for engineering problems, 1999.

[65] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proc. of the IEEE Intl. Conf. on Neural Networks*, pages 586–591, San Francisco, CA, 1993.

[66] Michael E. Rimer and Tony R. Martinez. Classification-based objective functions. *To appear in Machine Learning*, 2005.

[67] Steven Salzberg. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, 1(3):317–328, 1997.

[68] P.C. Sham and D. Curtis. An extended transmission/disequilibrium test (tdt) for multiallele marker loci. *Annals of Human Genetics*, 59(3):323–336, 1994.

[69] Jonathan R Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA, 1994.

[70] S. Stigler. Citation patterns in the journals of statistics and probability. *Statistical Science*, 9:94–108, 1994.

[71] The Entertainment Software Association. Essential facts about the computer and video game industry, 2006.

[72] Alex Waibel, S. Hidefumi, and K. Shikano. Consonant recognition by modular construction of large phonemic time-delay neural networks. In *Proceedings of the 6th IEEE International Conference on Acoustics, Speech, and Signal Processing.*, volume 1, pages 112–115, Glasgow, Scotland, 1989.

[73] Wikipedia. Elo rating system, 2006.

[74] David H. Wolpert and William G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, The Santa Fe Institute, Santa Fe, NM, 1995.

[75] Bruce Sterling Woodcock. Mmog active subscriptions 20.0 120,000+, 2006.

[76] K. Woods and K.W. Bowyer. Generating roc curves for artificial neural networks. *IEEE Transactions on Medical Imaging*, 16(3):329–337, 1997.

[77] B. Zadrozny. Reducing multiclass to binary by coupling probability estimates, 2001.

[78] Xinchuan Zeng and Tony Martinez. Using a neural networks to approximate an ensemble of classifiers. *Neural Processing Letters.*, 12(3):225–237, 2000.