Formalized Generalization Bounds for Perceptron-Like Algorithms

A thesis presented to

the faculty of

the Russ College of Engineering and Technology of Ohio University

In partial fulfillment

of the requirements for the degree

Master of Science

Robin J. Kelby

August 2020

This thesis titled

Formalized Generalization Bounds for Perceptron-Like Algorithms

by

ROBIN J. KELBY

has been approved for

the School of Electrical Engineering and Computer Science

and the Russ College of Engineering and Technology by

Gordon Stewart

Assistant Professor of Electrical Engineering and Technology

Mei Wei

Dean, Russ College of Engineering and Technology

Abstract

KELBY, ROBIN J., M.S., August 2020, Computer Science

Formalized Generalization Bounds for Perceptron-Like Algorithms (21 pp.)

Director of Thesis: Gordon Stewart

Insert your abstract here

Dedication

Dedicated to my Nathan.

Your patience, video games, and good cooking kept me going.

Acknowledgments

Acknowledge later

Table of Contents

List of Tables

Table Page

List of Figures

Figure Page

1 Introduction

The field of machine learning research has advanced very quickly in the past decade. Machine learning describes the class of computer programs that automatically learn from experience, often employed for classification, recognition, and clustering tasks. One of the classic problems in machine learning is handwritten digit recognition to classify written numbers automatically. Computers have historically struggled to interpret handwritten information because handwriting can vary drastically between writers. While humans can be taught to read as well as learn to read on their own, handwriting recognition can be challenging for computers to accomplish. Several datasets have been created specifically for the problem of handwriting analysis for numerical digits. For example, the MNIST dataset [LBBH98] is one of the primary datasets for computers to learn how to classify handwritten digits into the numbers 0-9. This dataset allows researchers to compare the performance of multiple models, trained and tested on the same data, but using different machine learning algorithms. Some systems have achieved a near-perfect performance on the MNIST dataset for the problem of handwritten digit classification, and this technology is valuable for processing documents, such as ZIP codes on letters sent through the U.S. Postal Service.

Increasingly, machine learning has been heavily integrated into our daily lives. As described by the authors of "Social media big data analytics", social media companies such as Facebook, Twitter, and YouTube learn from our digital data in order to serve individuals with targeted information and often advertising [GHHA19]. Retailers track customer purchases to learn about individuals' habits and entice them with specific offers and coupons. The pages we visit, profiles we create, and products we buy are used to predict our future actions and monetize our attention. This kind of task would be almost impossible for a human to complete, due to the vast amounts of data involved per person or account. In addition to social media, retailers, and advertisers, machine learning

techniques are also being employed in critical systems, such as healthcare and infrastructure, where failure can lead to the loss of time, money, and lives. Research to evaluate the use and oversight of machine learning algorithms [Var16] has shown that there are few existing safety principles and regulations for critical systems that rely on machine learning components. Machine learning drives more than websites and commerce; its algorithms are also responsible for the well-being and safety of people around the world, and regulation has largely not caught up with machine learning advances.

The development of machine learning tends to be experimentally driven in most applications. New or finely tuned configurations for internal components can lead to increased accuracy and efficiency or decreased training time compared to other algorithms for a specific problem or dataset. Such refinements can have enormous impacts for researchers studying machine learning problems and algorithms. The process of machine learning differs from algorithm to algorithm, but for most methods, machine learning algorithms learn models from training data to encode the program's knowledge. Models consist of learned parameters, which represent different kinds of data depending on the encoding of the model, and hyperparameters, a small number of variables directly specified by the programmer. Learned models are able to take a new piece of data as input and produce a result or judgment from that data. In the case of handwritten digits, the input to the model is the handwritten digit, and the output is the classification of that digit as a number from 0-9.

Machine learning algorithms can produce models that have millions of learned parameters, and small changes to model training, configuration, or hyperparameters can have enormous impacts on performance. Because of the complexity of the models produced by many machine learning algorithms, most new papers published in the field describe results found through experimentation, as opposed to examining the underlying

theory responsible for these advances. Additional research in understanding the theory behind machine learning may help to understand why some techniques are better suited for some problems than others, as well as potential avenues for exploration.

Finding errors in machine learning algorithms or models can be very difficult. With thousands or millions of parameters learned by the computer, not specified by the programmer, algorithms can easily get stuck in small, local solutions instead of finding the optimal solution. For example, gradient descent is an algorithm tasked with finding the lowest, or global, minimum of a multi-dimensional hillside with many peaks and valleys. Through many iterations, gradient descent travels downward along the gradient until a place is reached where descent is no longer possible. If the algorithm cannot find a deeper valley, this depth is returned as the overall solution. However, gradient descent can fail to find the global solution when the hyperparameters are not tuned correctly by the programmer or deeper valleys take too long to find. Techniques have been developed to mitigate the limitations of gradient descent, such as momentum, but the programmer usually has to experiment with multiple techniques to achieve peak performance. Additionally, few machine learning algorithms have theoretical properties that can be verified, such as a theorem that a learning algorithm will always terminate or find the global solution. Research into verifying machine learning to produce models with optimal behavior is limited due to these difficulties.

One way to increase our knowledge in the theory of machine learning is to verify the correctness of machine learning algorithms through mathematical proofs. Formal verification often entails machine-checked proofs of correctness, where software is built or translated into a proof assistant, such as Coq. Proof assistants allow for the integration of proofs with software specifications and implementations. Mathematical proofs in Coq are guaranteed to be as valid as the proof assistant itself, and because these proofs are portable programs, access to the proofs can allow others to verify proofs as well. Because

implementations are written in the same environment as their proofs, the proofs directly correspond to the implementation verified. The Coq environment also provides libraries containing both implementations of data structures and proofs to aid in the development of verified systems.

Researchers have used the Coq proof assistant to verify many different software systems and prove correctness properties. The CompCert compiler for the C language [Ler09] is the first verified compiler, proving that the behavior of a C program compiled with CompCert will not be changed in the transformation of compilation. Verified compilers ensure that the executable program produced by the compiler does not contain errors produced in compilation. For safety-critical applications, executables created by a verified compiler are more secure than executables created by unverified compilers. Another verified system written in Coq is Verdi [WWP+15], a framework for specifying and implementing distributed systems with tolerance for node faults. In a network of computers, connections can be dropped, packets lost or sent out of order, and nodes can fail or restart. Verdi allows the programmer to specify the fault conditions their distributed system should be resilient against, and the Verdi system itself mechanizes much of the proof process and code extraction for deployment in real-world networks. Distributed system software written with Verdi has been verified to handle faults and errors that may occur. Finally, Coq has also been used to implement microkernels, which are the basis for operating systems. The CertiKOS project [GSC+16] has developed several microkernels with security properties and proofs of correctness, including mC2, a verified concurrent microkernel. Operating systems allocate memory and computer resources and must defend against malicious processes. As demonstrated by these research projects, the Coq proof assistant can be extended for a diverse range of verified systems.

In this thesis, I will describe my additions to the verification framework MLCert. Building on the Perceptron implementation and existing proofs in MLCert, I present a

verified implementation of the Kernel Perceptron algorithm, as well as two variants on the Kernel Perceptron algorithm: a Budget Kernel Perceptron and a Description Kernel Perceptron. Background information for this thesis is provided in Chapter 2, with an introduction to the Perceptron and Kernel Perceptron algorithms, a more extended discussion of the challenges and tactics of machine learning verification, and the specifications for Budget Kernel Perceptrons and Description Kernel Perceptrons. Chapter 3 describes the methodology for implementing these algorithms in Coq. The proofs for these implementations and their performance results are detailed in Chapter 4. Finally, future work and conclusions are discussed in Chapter 5.

2 Background

This chapter aims to provide necessary background information in order to understand the remainder of this thesis. The first two sections describe the Perceptron algorithm and its descendant, the Kernel Perceptron algorithm. Next, the challenges and methods of formal verification of machine learning are discussed in sections 2.3 and 2.4. Finally, modifications of the Kernel Perceptron algorithm, such as Budget Kernel Perceptrons in section 2.5 and Description Kernel Perceptrons in section 2.6, are detailed as improvements for the Kernel Perceptron.

2.1 The Perceptron Algorithm

The Perceptron algorithm was initially published in 1957 by Frank Rosenblatt. Highly influential in the early growth and development of the field of artificial intelligence, the Perceptron [Ros57] provided one of the first methods for computers to iteratively learn to classify data into discrete categories. In order to classify n-dimensional data, the Perceptron learns a weight vector with n parameters as well as a bias term. Both the weight vector and bias consist of positive integers greater than or equal to zero which encode a linear hyperplane separating two or more categories in n-dimensional space.

The most basic Perceptron algorithm has the following steps. Before training, each parameter in the weight vector w is initialized to zero. The algorithm consists of two nested loops, as shown by the following pseudocode:

For this algorithm, we require the weight vector, the number of epochs, and the training set as input. The training set consists of labeled training examples, where the label is either 0 or 1. The outer loop uses the number of epochs to control the number of iterations over the entire training set. The inner loop executes for every training example in the training set and has two main steps. First, the n-dimensional data inside the training example and the weight vector are used to calculate the Perceptron's predicted label for

this example, without using the training example's true label. The equation for Perceptron prediction takes as input the weight vector and a single training example to produce a predicted label for this example:

$$predict = bias + (w \cdot example) \tag{2.1}$$

The true label and the calculated predicted label are then compared. If both labels are the same, the Perceptron correctly classified this training example. However, if the predicted label is different, the weight vector is updated using the example to improve classification over time. This update is the second step of the inner loop.

There are limitations to the Perceptron's classification. The Perceptron cannot classify data that is not linearly separable with 100% accuracy, such as points classified by the exclusive-OR function, a binary operator that returns TRUE when its two inputs are the opposite of each other. Despite the simplicity of exclusive-OR, the Perceptron cannot produce a model, or linear hyperplane, such that all the points classified by exclusive-OR as TRUE are also classified by the Perceptron as TRUE, and all the points classified by exclusive-OR as FALSE are also classified by the Perceptron as FALSE. The Perceptron can achieve at best 75% accuracy for the exclusive-OR function. This restriction on the Perceptron caused the first AI Winter, a severe decline in artificial intelligence research, due to unreasonable expectations for the Perceptron in fields where data is not linearly separable.

While the Perceptron is limited to classification of linearly separable data, the Perceptron Convergence Theorem states that the Perceptron is guaranteed to converge to a solution on linearly separable data. This property of the Perceptron algorithm was first proven on paper by Papert in 1961 [Pap61] and Block in 1962 [Blo62]. However, this property was proven sound by Murphy, Gray, and Stewart [MGS17] with a proof of

correctness written in the Coq proof assistant, to our knowledge the first time this proof was implemented in a proof assistant and verified by machine.

2.2 The Kernel Perceptron

3 Methods

4 Results

5 Conclusions

[MGS17] [ABR64] [LTS90] [CCBG07] [DSSS07] [CKS03] [OKC09] [BS19] [BF16] [TD05]

References

[ABR64] M. A. Aizerman, E. M. Braverman, and L. I. Rozoner. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

[BF16] Adrien Bibal and Benoit Frénay. Interpretability of machine learning models and representations: an introduction. In *ESANN'16 Proceedings*, Bruges, 2016.

[Blo62] Hans-Dieter Block. The perceptron: a model for brain functioning. *Reviews of Modern Physics*, 34(1):123, 1962.

[BS19] Alexander Bagnall and Gordon Stewart. Certifying the true error: machine learning in coq with verified generalization guarantees. In *Proceedings of AAAI'19*, pages 2662–2669, Hawaii, 2019.

[CCBG07] Giovanni Cavallanti, Nicolo Cesa-Bianchi, and Claudio Gentile. Tracking the best hyperplane with a simple budget perceptron. *Machine Learning*, 69(23):143–167, 2007.

[CKS03] Koby Crammer, Jaz Kandola, and Yoram Singer. Online classification on a budget. In *Advances in Neural Information Processing Systems 16*. Proceedings of NIPS 2003, 2003.

[DSSS07] Ofer Dekel, Shai Shalev-Shwartz, and Yoram Singer. The forgetron: a kernel-based perceptron on a budget. *SIAM Journal on Computing*, 37(5):1342–1372, 2007.

[GHHA19] Norjihan A. Ghani, Suraya Hamid, Ibrahim A. T. Hashem, and Ejaz Ahmed. Social media big data analytics. *Computers in Human Behavior*, 101:417–428, 2019.

[GSC+16] Ronghui Gu, Zhong Shao, Hao Chen, Xiongnan (Newman) Wu, Jieung Kim, Vilhelm Sjoberg, and David Costanzo. Certikos: an extensible architecture for building certified concurrent os kernels. In *OSDI'16*, pages 653–669, Savannah, Georgia, 2016.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86(11), pages 2278–2324, 1998.

[Ler09] Xavier Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 2009.

[LTS90] Esther Levin, Naftali Tishby, and S. A. Solla. A statistical approach to learning and generalization in layered neural networks. In *Proceedings of the IEEE*, volume 78, pages 1568–1574, 1990.

[MGS17] Charlie Murphy, Patrick Gray, and Gordon Stewart. Verified perceptron convergence theorem. In *MAPL'17*, Barcelona, 2017.

[OKC09] Francesco Orabona, Joseph Keshet, and Barbara Caputo. Bounded kernel-based online learning. *Journal of Machine Learning Research*, 10(11):2643–2666, 2009.

[Pap61] Seymour Papert. Some mathematical models of learning. In *Proceedings of the Fourth London Symposium on Information Theory*, 1961.

[Ros57] Frank Rosenblatt. The perceptron, a perceiving and recognizing automaton. *Report: Cornell Aeronautical Laboratory*, 58(460), 1957.

[TD05] Brian. J. Taylor and Marjorie A. Darrah. Rule extraction as a formal method for the verification and validation of neural networks. In *Proceedings of IEEE International Joint Conference on Neural Networks 2005*, pages 2915–2920, Montreal, 2005.

[Var16] Kush R. Varshney. Engineering safety in machine learning. In *2016 Information Theory and Applications Workshop*, La Jolla, California, 2016.

[WWP+15] James R. Wilcox, Doug Woos, Pavel Panchekha, Zachary Tatlock, Xi Wang, Michael D. Ernst, and Thomas Anderson. Verdi: A framework for implementing and formally verifying distributed systems. In *PLDI'15*, pages 357–368, Portland, Oregon, 2015.