The Dissertation Committee for Santiago Gonzalez
certifies that this is the approved version of the following dissertation:

Improving Deep Learning Through Loss-Function Evolution

Committee:

Risto Miikkulainen, Supervisor

Wolfgang Banzhaf

Greg Durrett

Qixing Huang

Improving Deep Learning Through Loss-Function Evolution

by

Santiago Gonzalez

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2020

Dedicated to the men and women of science.

Acknowledgments

I am grateful for all the people that have made this work possible. First, I am thankful for Dr. Risto Miikkulainen for taking a chance on me, for being a great source of wisdom, feedback, and expertise, and for teaching me how to weave a story through even the most abstract material. I would also like to thank Dr. Tracy Camp for having taught me how to conduct rigorous scientific research and write academic papers.

My thanks also go out to the dissertation committee members, whose feedback and thoughts helped shape certain directions of my work.

I would like to thank Karl Mutch and Sid Stuart for their help working with Studio and infrastructure; Mohak Kant for his work integrating TaylorGLO into LEAF and running TaylorGLO GAN experiments; Babak Hodjat, Hormoz Shahrzad, and Elliot Meyerson for their engaging conversations on evolution; and the rest of the Cognizant Evolutionary AI team.

I am thankful to my parents and sister Andrea for always being there for me throughout this eventful journey and for their moral support and guidance.

Thank you to Clayton Sanford for his conversations and thoughts around learning theory and generalization and his pointers to insightful papers.

I would like to thank Katie Traughber Dahm and the rest of the administrative staff at UT Austin for their kindness and help navigating univer-

sity processes. Finally, I am thankful for the United States National Security Agency (NSA), through GFSD, and the University of Texas at Austin for their generosity.

Improving Deep Learning Through Loss-Function Evolution

Santiago Gonzalez, Ph.D.
The University of Texas at Austin, 2020

Supervisor: Risto Miikkulainen

As the complexity of neural network models has grown, it has become increasingly important to optimize their design automatically through metalearning. Methods for discovering hyperparameters, topologies, and learning rate schedules have lead to significant increases in performance. This dissertation tackles a new type of metalearning: loss-function optimization. Loss functions define a model's core training objective and thus present a clear opportunity. Two techniques, GLO and TaylorGLO, were developed to tackle this metalearning problem using genetic programming and evolutionary strategies. Experiments show that neural networks trained with metalearned loss functions are more accurate, have higher data utilization, train faster, and are more robust against adversarial attacks. A theoretical framework was developed to analyze how and why different loss functions bias training towards different regions of the parameter space. Using this framework, their performance gains are found to result from a regularizing effect that is tailored to each domain. Overall, this dissertation demonstrates that new, metalearned

loss functions can result in better trained models, and provides the next stepping stone towards fully automated machine learning.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Machine learning and artificial intelligence have led to the modernization of many industries and to the creation of novel systems that have the potential to better humanity. Large increases in compute power have made it possible to train models of increasing sophistication. However, such models can be overwhelmingly complex: they have many nonintuitively interacting components, and there is no rigourous theory on how they work. As a result, the field is less scientific and methodical and depends more on human intuition. The ability to reason about model complexity has exceeded the limits of most humans, impeding progress, requiring special, hard-to-find expertise, and increasing the cost of developing models.

Consider a better future: Models are able to automatically adapt to best serve their target domain. The field of metalearning aims to reach this future by optimizing various parts of machine learning automatically. This dissertation explores a new avenue towards this goal: loss-function metalearning. By automatically and methodically searching the space of loss functions, model performance can be improved without manual tuning. Metalearned loss functions establish a form of regularization, biasing the training process

towards more robust configurations.

In this dissertation, two techniques are developed for this purpose: *Genetic Loss-function Optimization* (GLO), an evolutionary approach that optimizes loss functions as trees with coefficients, maximizing creativity; and *Multivariate Taylor expansion-based genetic loss-function optimization* (TaylorGLO), an alternative technique that scales well to models with millions of trainable parameters.

1.1 Motivation

Much of the power of modern neural networks originates from their complexity, i.e., number of parameters, hyperparameters, and topology. This complexity is often beyond human ability to optimize, and automated methods are needed. An entire field of metalearning has emerged recently to address this issue, based on various methods such as gradient descent, simulated annealing, reinforcement learning, Bayesian optimization, and evolutionary computation (EC) [38].

Metalearning has repeatedly demonstrated how various aspects of machine learning can be automatically tuned without human intervention. For example, neural architecture search, i.e. metalearning of model architectures, has resulted in models that outperform those designed by humans [115, 136, 164]. Many of these techniques use evolution and discover solutions with unique architectural motifs.

While a wide repertoire of work now exists for optimizing many aspects of neural networks, the dynamics of training are still usually set manually without a concrete, scientific principle. It is important, however: training schedules, loss functions, and learning rates all affect the training and final functionality of a neural network. Perhaps they could also be optimized through metalearning? This dissertation verifies this hypothesis, focusing on the optimization of loss functions.

Loss functions are the fundamental guide for the training process, serving as the root goal against which a model's trainable parameters are optimized. The choice of loss function implicitly biases the optimization process [19], i.e. the way learning and generalization occur.

In modern machine learning, however, most practitioners rely on a few standard loss functions. For example, in classification settings, the cross-entropy loss is almost exclusively used. Considering that seemingly small differences can have a great impact on a model's performance (e.g., the ordering of layers [72]), it is suprising that such a fundamental aspect of training is usually not considered as part of model design.

Loss-function metalearning can thus serve as a way to find appropriate regularization for neural networks automatically, resulting in models with better performance.

1.2 Challenges

Theory has lagged behind practice in deep learning since its inception, and the same is particularly true for regularization. While the past several years have resulted in many theoretical findings on regularization, they tend to focus on shallow multi-layer perceptrons and specific scenarios that are not representative of real-world use-cases.

As a result, practitioners have used a handful of different loss functions, such as the cross-entropy loss, across wide varieties of problems. Justifications for different loss functions have typically been speculative. For example, probabilistic intuitions are sometimes used to justify the cross-entropy loss in classification settings; however, modern neural network outputs should *not* be interpreted as probabilities [45].

Since optimal loss functions cannot yet be designed from first principles, a search based method must be used. At first, gradient-based optimization techniques may seem like an appropriate option, given their ubiquity in modern machine learning. However, since many neural network objectives (e.g., accuracy) are inherently non-differentiable, gradient-based approaches are not always possible. This observation leads to evolutionary computation, which can tackle problems with rugged fitness landscapes that need not be differentiable.

Even if evolutionary computation is used for optimization, loss functions need to be represented in a way that makes it possible to both encode

a variety of functions and evolve them effectively. This dissertation presents two representations that lie at different points along the continuum between representational flexibility and efficiency.

Search based approaches, including evolutionary computation, require a way to evaluate the efficacy of candidates. That is, given a loss function, how well does a model trained with it perform? Given the limited theoretical understanding in deep learning, it can be challenging to predict the outcome that a particular loss function will have on training. Thus, models must be trained with each candidate. Since the dynamics of training are not constant throughout the training process, evaluations based on partial training are noisy. Significant amounts of compute is necessary, reducing the number of candidates that can be evaluated.

Further, modern deep learning has been inadvertently designed and optimized for a few loss functions, such as cross-entropy. If other aspects of the model are kept fixed, the benefits of loss-function metalearning may be underestimated. Nonetheless, the loss-function metalearning techniques in this disssessrtation are able to train network that outperform those trained with cross-entropy.

1.3 Approach

A general framework for loss function metalearning, covering both novel loss function discovery and optimization, is developed and evaluated experimentally. This framework, Genetic Loss-function Optimization (GLO), lever-

ages Genetic Programming to build loss functions represented as trees. These functions are repeatedly recombined and mutated to find an optimal structure, and subsequently a Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is used to optimize their coefficients.

EC methods were chosen because EC is arguably the most versatile of the metalearning approaches. EC, being a type of population-based search method, allows for extensive exploration, which often results in creative, novel solutions [100]. EC has been successful in hyperparameter optimization and architecture design in particular [108, 115, 136, 164]. It has also been used to discover mathematical formulas to explain experimental data [155]. It is, therefore, likely to find creative solutions in the loss-function optimization domain as well.

Networks trained with GLO loss functions are found to outperform the cross-entropy loss on standard image classification tasks. Training with these new loss functions requires fewer steps, results in lower test error, and allows for smaller datasets to be used. Indeed, on the MNIST image classification benchmark, GLO discovered a surprising new loss function, named Baikal for its shape. This function performs very well, presumably by establishing an implicit regularization effect. Baikal outperforms the standard cross-entropy loss in terms of training speed, final accuracy, and data requirements. Furthermore, Baikal was found to transfer these benefits to a more complicated classification task, CIFAR-10.

At first glance, Baikal behaves rather unintuitively; loss does not de-

crease monotonically as a network's predictions become more correct. Upon further analysis, Baikal was found to perform implicit regularization. By preventing the network from being too confident in its predictions, training with Baikal produced a more robust model. This finding was surprising and encouraging, since it means that GLO is able to discover loss functions that train networks that are more generalizable and overfit less.

While GLO's tree-based representation gives it unbounded representational flexibility, this flexibility can make evolution a challenge—requiring large populations evolved over many generations—since the vast majority of functions in such a space are not, in fact, usable as loss functions. Additionally, GLO's two-phased approach to loss function discovery and optimization can result in loss functions that are greedily, rather than mutually, optimized for each phase. That is, a function's structure may only be optimal with a certain set of coefficients.

Aiming to resolve these shortcomings, another variation of GLO was developed. The new technique, Multivariate Taylor expansion-based genetic loss-function optimization (TaylorGLO), uses a new representation for loss functions based upon multivariate function approximators, particularly Taylor expansions. This parameterization helps combine the previously separate discovery of structure and optimization of coefficients into one step, while simultaneously providing a well-behaved search space. While this representation is more rigid than the tree-based representation, and may not result in solutions that are as creative, they are often more practical, and can be applied

reliably to many real-world use cases.

TaylorGLO was used to evolve loss functions for a wide variety of modern architectures on four different benchmark datasets. Networks trained with TaylorGLO loss functions consistently outperform those trained with cross-entropy loss. TaylorGLO also improves performance of networks trained with other regularization techniques, suggesting that it provides a complementary form or regularization.

By fundamentally altering the training process, TaylorGLO loss functions allow a model's parameters to reach unique regions of the trainable parameter space. It results in models with flatter minima, i.e. models' outputs are less sensitive to small changes in the model's trainable parameters, than those trained with cross-entropy loss. As a result, TaylorGLO models are more robust and generalize better.

In order to characterize this process, a new theoretical framework for reasoning about the optimization biases and regularization effects imparted by loss functions is developed. A decomposition process can be applied to the standard stochastic gradient descent learning rule to get expressions that describe a loss function's behavior uniquely. These behaviors are analyzed in different settings, showing how metalearned loss functions bias and regularize training implicitly. A further analysis across the totality of training shows how loss functions encourage or discourage fitting to individual samples. Two specific metalearned loss functions, one evolved with GLO and one with TaylorGLO, are found to encourage the model to fit misclassified samples and

those with moderately confident classifications, while penalizing overly confident predictions. The cross-entropy and mean-squared-error loss functions, conversely, encourage data fitting in all cases, resulting in overfitting.

TaylorGLO is further extended to four special settings, demonstrating its ability to leverage previously gained insights in machine learning. First, TaylorGLO is used to evolve multiple, separate loss functions for models with auxiliary classifiers, impacting different parts of a model in different ways. Second, the TaylorGLO search process is improved using an invariant on loss function parameters that drives it towards more useful candidates more effectively. Third, TaylorGLO loss functions are found to make networks more robust against adversarial attacks, and an additional adversarial objective further increases this effect. Fourth, TaylorGLO is leveraged to discover new formulations for conditional generative adversarial networks that are able to generate higher quality images.

Thus, GLO and TaylorGLO together constitute a comprehensive first foray into loss-function metalearning. GLO provides a high-level of flexibility in evolving loss functions with many shapes and many inputs, while TaylorGLO evolves loss functions for deep neural networks effectively. They allow practitioners to metalearn regularization that is customized to individual tasks automatically, resulting in higher-performing models that are more robust. This dissertation also takes first steps in theoretical analysis of regularization and loss functions, providing a stepping stone towards further, more principled work on metalearning.

1.4 Guide to the Reader

The disposition of chapters in this dissertation is as follows:

Chapter 2 reviews background concepts and relevant literature in the field. Deep learning, evolution, loss functions, and regularization are detailed, motivating the development of loss-function metalearning techniques, and informing their design.

Chapter 3 describes the experimental setup and methodology for the remainder of the dissertation. Described therein are the datasets and neural network architectures used to evaluate techniques, the novel experiment management system called Lossferatu that was built for this purpose, and statistical techniques that help illustrate results.

Chapter 4 introduces Genetic Loss-function Optimization (GLO), the first, flexible technique for loss-function metalearning. The technique is evaluated experimentally and its merits and shortcomings are discussed.

Chapter 5 describes TaylorGLO, a more practical technqiue for loss-function metalearning. Building upon the lessons learned from GLO, TaylorGLO is evaluated empirically on a variety of deep learning settings. Its loss functions result in fundamentally different trained networks, thanks to a qualitatively different training process.

Chapter 6 develops a theoretical framework to characterize the regularization processes that result in evolved loss functions. The theoretical insights are connected back to empirical results.

Chapter 7 delves into four opportunities to extend: auxiliary classifier loss functions, an invariant on TaylorGLO parameters, evolving loss functions that are hardened against adversarial attacks, and evolving loss functions for generative adversarial networks.

Chapter 8 summarizes the findings in this dissertation, discusses the observed interactions between TaylorGLO and other regularization techniques, proposes future work, and explores the broader impact of this work.

Chapter 9 closes with an overview of the impact of this dissertation and its contributions to science. The ultimate conclusion is that loss-function metalearning is a useful new subfield that can improve deep models through learned regularization.

Chapter 2

Background

This chapter reviews the key areas that motivate, provide a foundation, and help shape the dissertation's work. This review begins with deep learning and the role of loss functions. Next, generative adversarial networks (GANs)—a set of techniques that leverage deep learning for generation tasks—and their loss functions are presented, highlighting key developments and challenges. Subsequently, evolutionary computation, the foundation for the techniques developed in this dissertation, is covered. This review leads to a discussion of metalearning literature and how evolution fits into it, particularly in terms of neural architecture search (NAS). Finally, regularization and optimization biases in modern machine learning are evaluated.

2.1 Deep Learning and Loss Functions

Interest in machine learning and AI has recently risen thanks to modern deep learning [98]. While deep learning has varying meanings, within this body of work, it is used to refer to neural networks of a higher depth and complexity than traditional multilayer perceptrons [142]; i.e., networks that learn internal data representations of increasing abstraction. As illustrated in

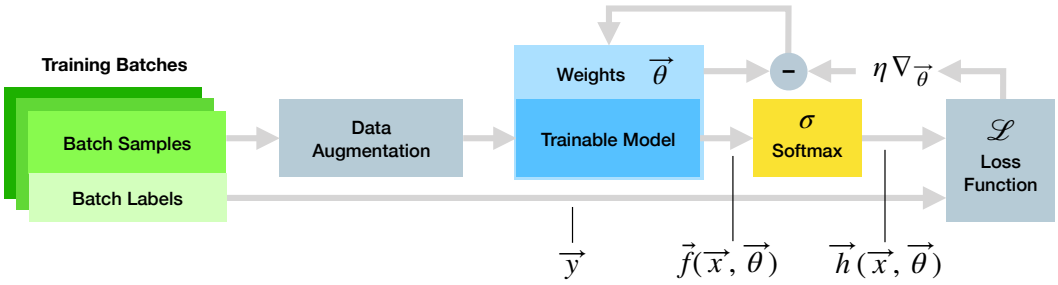


Figure 2.1: **Neural network training by gradient descent in a classification setting.** A training dataset is sampled into batches that that may undergo data augmentation prior to being passed through the model. The model's outputs and labels are inputs to a loss function, from which gradients are calculated to update the model's trainable parameters. Metalearning aims to optimize parts of this process automatically to result in trained models with better performance.

Figure 2.1, deep models, and neural networks overall, are traditionally trained iteratively, whereby model parameters (i.e., weights and biases, at a minimum) are updated using gradients that are propagated backwards through the network, starting from an error given by a loss function [143]. Loss functions represent the primary training objective of a neural network.

For many tasks, such as classification and language modeling, the cross-entropy loss (also known as the log loss) is used almost exclusively. Note that while regularization terms are sometimes added to this definition, the core component is still the cross-entropy loss. Information-theoretic reasoning is used to motivate the log loss: It aims to minimize the number of bits needed to identify a message from the true distribution, using a code from the predicted distribution. More specifically, within a classification task, the log loss is

defined as

$$\mathcal{L}_{\text{Log}} = -\frac{1}{n}\sum_{i=0}^{n} x_i \log(y_i) \ , \tag{2.1}$$

where $\mathbf{x}$ is sampled from the true distribution, $\mathbf{y}$ is sampled from the predicted distribution, and n is the number of classes. This notation is used in subsequent chapters to describe all loss functions.

While the variety in loss functions for classification is rather low, different types of tasks that do not fit neatly into a single-label classification modality often have different loss functions. Data reconstruction tasks and autoencoders often may use an L_2 loss (also referred to in the literature as the squared-difference similarity or mean-squared error) [47, 51]. Kullback-Leibler divergence can be used as a loss when comparing the similarity of two distributions [91, 200]. For building visual-semantic text embeddings, a pair-wise ranking loss based on the cosine similarity between images and fragments of text can be used [36]. Tasks that can be represented as binary classification tasks, such as true/false question answering or paraphrase identification, often use the binary case of the cross-entropy loss [200].

Every instance where a loss function is chosen for a task without a specific justification is an opportunity to find a more optimal loss function automatically. The choice of loss function can have a great impact on performance, as will be seen in this dissertation. A particularly interesting case is training generative adversarial networks, where loss functions play central role, as will be reviewed next.

2.2 Generative Adversarial Networks

Generative Adversarial Networks (GANs), initially conceived by Goodfellow et al. [55], are a type of generative model consisting of a pair of networks, a generator and discriminator, that are trained in tandem. GANs are a modern successor to Variational Autoencoders (VAEs) [91] and Boltzmann Machines [74], including Restricted Boltzmann Machines [159] and Deep Boltzmann Machines [149].

The following subsections review prominent GAN methods. Key GAN formulations, and the relationships between them, are described. Consistent notation (shown in Table 2.1) is used, consolidating the extensive variety of notation in the field.

Table 2.1: GAN Notation Decoder

Symbol	**Description**
$G(\boldsymbol{x}, \theta_G)$	Generator function
$D(\boldsymbol{z}, \theta_D)$	Discriminator function
$\mathbb{P}_{\text{data}}$	Probability distribution of the original data
$\mathbb{P}_z$	Latent vector noise distribution
$\mathbb{P}_g$	Probability distribution of $G(\boldsymbol{z})$
$\boldsymbol{x}$	Data, where $\boldsymbol{x} \sim \mathbb{P}_{\text{data}}$
$\tilde{\boldsymbol{x}}$	Generated data
$\boldsymbol{z}$	Latent vector, where $\boldsymbol{z} \sim \mathbb{P}_z$
$\boldsymbol{c}$	Condition vector
λ	Represents various types of weights / hyperparameters

2.2.1 Overview

A GAN's generator and discriminator are set to compete with each other in a minimax game, attempting to reach a Nash equilibrium [73, 123]. Throughout the training process, the generator aims to transform samples from a prior noise distribution into data, such as an image, that tricks the discriminator into thinking it has been sampled from the real data's distribution. Simultaneously, the discriminator aims to determine whether a given sample came from the real data's distribution, or was generated from noise.

Unfortunately, GANs are difficult to train, frequently exhibiting instability, i.e., mode collapse, where all modes of the target data distribution are not fully represented by the generator [10, 61, 85, 111, 112, 114, 135]. GANs that operate on image data often suffer from visual artifacts and blurring of generated images [85, 130]. Additionally, datasets with low variability have been found to degrade GAN performance [111].

GANs are also difficult to evaluate quantitatively, typically relying on metrics that attempt to embody vague notions of quality. Popular GAN image scoring metrics, for example, have been found to have many pitfalls, including cases where two samples of clearly disparate quality may have similar values [20].

2.2.2 Original Minimax and Non-Saturating GAN

Using the notation described in Table 2.1, the original minimax GAN formulation from [55] can be defined as

$$\min_{\theta_G} \max_{\theta_D} \; \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_{\text{data}}} [\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_z} [\log (1 - D(G(\boldsymbol{z})))] \; . \tag{2.2}$$

This formulation can be broken down into two separate loss functions, one each for the discriminator and generator:

$$\mathcal{L}_D \;=\; -\frac{1}{n}\sum_{i=1}^{n} [\log D(x_i) + \log(1 - D(G(z_i)))] \text{ , and} \tag{2.3}$$

$$\mathcal{L}_G \;=\; \frac{1}{n}\sum_{i=1}^{n} \log(1 - D(G(z_i))) \; . \tag{2.4}$$

The discriminator's loss function is equivalent to a sigmoid cross-entropy loss when thought of as a binary classifier. Goodfellow et al. [55] proved that training a GAN with this formulation is equivalent to minimizing the Jensen-Shannon divergence between $\mathbb{P}_g$ and $\mathbb{P}_{\text{data}}$, i.e. a symmetric divergence metric based on the Kullback-Leibler divergence.

In the above formulation the generator's loss saturates quickly since the discriminator learns to reject the novice generator's samples early on in training. To resolve this problem, Goodfellow et al. provided a second "non-saturating" formulation with the same fixed-point dynamics, but better, more intense gradients for the generator early on:

$$\max_{\theta_D} \quad \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_{\text{data}}} [\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_z} [\log (1 - D(G(\boldsymbol{z})))] \; , \tag{2.5}$$

$$\max_{\theta_G} \quad \mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_z} [\log D(G(\boldsymbol{z}))] \; . \tag{2.6}$$

Each GAN training step consists of training the discriminator for k steps, while sequentially training the generator for only one step. This difference in steps for both networks helps prevent the discriminator from learning too quickly and overpowering the generator.

Alternatively, Unrolled GANs [114] aimed to prevent the discriminator from overpowering the generator by using a discriminator which has been unrolled for a certain number of steps in the generator's loss, thus allowing the generator to train against a more optimal discriminator. More recent GAN work instead uses a two time-scale update rule (TTUR) [73], where the two networks are trained under different learning rates for one step each. This approach has proven to converge more reliably to more desirable solutions.

Unfortunately, with both minimax and non-saturating GANs the generator gradients vanish for samples that are on the correct side of the decision boundary but far from the true data distribution [111, 112]. The Wasserstein GAN, described next, is designed to solve this problem.

2.2.3 Wasserstein GAN

The Wasserstein GAN (WGAN) [10] is arguably one of the most impactful developments in the GAN literature since the original formulation by Goodfellow et al. [55]. WGANs minimize the Wasserstein-1 distance between $\mathbb{P}_g$ and $\mathbb{P}_{\text{data}}$, rather than the Jensen-Shannon divergence, in an attempt to avoid vanishing gradient and mode collapse issues. In the context of GANs,

the Wasserstein-1 distance can be defined as

$$W(\mathbb{P}_g, \mathbb{P}_{\text{data}}) = \inf_{\gamma \in \Pi(\mathbb{P}_g, \mathbb{P}_{\text{data}})} \mathbb{E}_{(\boldsymbol{u},\boldsymbol{v})\sim\gamma} \left[\|\boldsymbol{u} - \boldsymbol{v}\|\right] \ , \tag{2.7}$$

where, $\gamma(\boldsymbol{u}, \boldsymbol{v})$ represents the amount of mass that needs to move from $\boldsymbol{u}$ to $\boldsymbol{v}$ for $\mathbb{P}_g$ to become $\mathbb{P}_{\text{data}}$. This formulation with the infimum is intractable, but the Kantorovich-Rubinstein duality [182] with a supremum makes the Wasserstein-1 distance tractable, while imposing a 1-Lipschitz smoothness constraint:

$$W(\mathbb{P}_g, \mathbb{P}_{\text{data}}) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{\boldsymbol{u}\sim\mathbb{P}_g} \left[f(\boldsymbol{u})\right] - \mathbb{E}_{\boldsymbol{u}\sim\mathbb{P}_{\text{data}}} \left[f(\boldsymbol{u})\right] \ , \tag{2.8}$$

which translates to the training objective

$$\min_{\theta_G} \max_{\theta_D \in \Theta_D} \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_{\text{data}}} \left[D(\boldsymbol{x})\right] - \mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_z} \left[D(G(\boldsymbol{z}))\right] \ , \tag{2.9}$$

where Θ_D is the set of all parameters for which D is a 1-Lipschitz function.

WGANs are an excellent example of how generator and discriminator loss functions can profoundly impact the quality of generated samples and the prevalence of mode collapse. However, the WGAN has a 1-Lipschitz constraint that needs to be maintained throughout training for the formulation to work. WGANs enforce the constraint via gradient clipping, at the cost of requiring an optimizer that does not use momentum, i.e., RMSProp [172] rather than Adam [92].

To resolve the issues caused by gradient clipping, a subsequent formulation, WGAN-GP [61], added a gradient penalty regularization term to the

discriminator loss:

$$GP = \lambda \, \mathbb{E}_{\hat{\boldsymbol{x}} \sim \mathbb{P}_{\hat{x}}} \left[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2 \right] \ , \tag{2.10}$$

where $\mathbb{P}_{\hat{x}}$ samples uniformly along lines between $\mathbb{P}_{\text{data}}$ and $\mathbb{P}_g$. The gradient penalty enforces a soft Lipschitz smoothness constraint, leading to a more stationary loss surface than when gradient clipping is used, which in turn makes it possible to use momentum-based optimizers. The gradient penalty term has even been successfully used in non-Wasserstein GANs [39, 111]. However, gradient penalties can increase memory and compute costs [111].

2.2.4 Least-Squares GAN

Another attempt to solve the issue of vanishing gradients is the Least-Squares GAN (LSGAN) [112]. It defines the training objective as

$$\min_{\theta_D} \quad \frac{1}{2} \, \mathbb{E}_{\boldsymbol{x} \sim \mathbb{P}_{\text{data}}} \left[(D(\boldsymbol{x}) - b)^2 \right] + \mathbb{E}_{\boldsymbol{z} \sim \mathbb{P}_z} \left[(D(G(\boldsymbol{z})) - a)^2 \right] \ , \tag{2.11}$$

$$\min_{\theta_G} \quad \frac{1}{2} \, \mathbb{E}_{\boldsymbol{z} \sim \mathbb{P}_z} \left[(D(G(\boldsymbol{z})) - c)^2 \right] \ , \tag{2.12}$$

where a is the label for generated data, b is the label for real data, and c is the label that G wants to trick D into believing for generated data. In practice, typically $a = 0, b = 1, c = 1$. However, subsequently, $a = -1, b = -1, c = 0$ were found to result in faster convergence, making it the recommended parameter setting [61]. Training an LSGAN was shown to be equivalent to minimizing the Pearson χ^2 divergence [132] between $\mathbb{P}_{\text{data}} + \mathbb{P}_g$ and $2 * \mathbb{P}_g$. Generated data quality can oscillate throughout the training process [111], indicating a disparity between data quality and loss.

2.2.5 InfoGAN

InfoGAN [27], derived from information theory, attempts to learn a disentangled representation for the latent space. This goal is accomplished through a term that is subtracted from the training objective. With a coefficient λ, it bounds the mutual information between elements in the latent space and generated samples:

$$\mathcal{L}_I(\boldsymbol{d}, G) = \mathbb{E}_{\tilde{\boldsymbol{x}} \sim G(\boldsymbol{z} \oplus \boldsymbol{d}), \boldsymbol{z} \sim \mathbb{P}_z} \left[\mathbb{E}_{\boldsymbol{d}' \sim P(\boldsymbol{d}|\tilde{\boldsymbol{x}})} \left[\log Q(\boldsymbol{d}'|\tilde{\boldsymbol{x}}) \right] \right] + H(\boldsymbol{d}) \leq I(\boldsymbol{d}; G(\boldsymbol{z} \oplus \boldsymbol{d})) \ . \tag{2.13}$$

This term results in variational regularization of mutual information [12]. The latent vector is broken into the traditional noise component $\boldsymbol{z}$ and a latent code component $\boldsymbol{d}$, which are concatenated. The latent code entropy, $H(\boldsymbol{d})$ is treated as a constant. The auxiliary posterior Q, which aims to approximate $P(\boldsymbol{d}|\boldsymbol{x})$, is based on D.

InfoGANs can learn latent codes on several different image datasets successfully. However, the size and sampling distribution of the latent code must be defined *a priori*, thus requiring manual design. On the other hand, it is thusu possible to steer the latent code conceptually. For example, if the data's sampling distribution is known *a priori* to have ten modes, an element of the latent code could be $d_i \sim \text{Categorical}(k = 10, p = 0.1)$.

2.2.6 Conditional GAN

Traditional GANs learn how to generate data from a latent space, i.e. an embedded representation of the training data that the generator constructs.

Typically, the elements of a latent space have no immediately intuitive meaning [27, 97]. Thus, GANs can generate novel data, but there is no way to steer the generation process to generate particular types of data. For example, a GAN that generates images of human faces cannot be explicitly told to generate a face with a particular hair color or of a specific gender. While techniques have been developed to analyze this latent space [103, 183], or build more interpretable latent spaces during the training process [27], they do not necessarily translate a human's prior intuition correctly or make use of labels when they are available. To tackle this problem, Conditional GANs, first proposed as future work in [55] and subsequently developed by Mirza and Osindero [117], allow directly targetable features (i.e., conditions) to be an integral part of the generator's input.

The conditioned training objective for a minimax GAN can be defined, without loss of generality, as

$$\min_{\theta_G} \max_{\theta_D} \; \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_{\text{data}}} \left[\log D(\boldsymbol{x} \oplus \boldsymbol{c})\right] + \mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_z} \left[\log\left(1 - D(G(\boldsymbol{z} \oplus \boldsymbol{c}))\right)\right] \;, \quad (2.14)$$

where $\boldsymbol{z} \oplus \boldsymbol{c}$ is basic concatenation of vectors. During training, the condition vector, $\boldsymbol{c}$, arises from the sampling process that produces each $\boldsymbol{x}$. This same framework can be used to design conditioned variants of other GAN formulations.

Conditional GANs have enjoyed great successes as a result of their flexibility, even in the face of large, complex condition vectors, which may even be whole images. They enable new applications for GANs, including image

to image translation [85], the generation of images given text [139], repairing software vulnerabilities (framed as sequence to sequence translation) [69], and integrated circuit mask design [9]. Notably, conditional GANs can increase the quality of generated samples for labeled datasets, even when conditioned generation is not needed [179].

2.2.7 Opportunity: Optimizing Loss Functions

Overall, competing GAN formulations all have one property in common: the generator and discriminator loss functions have been arduously derived by hand. GAN performance and stability is greatly impacted by the choice of loss functions. Different regularization terms, such as the aforementioned gradient penalty can also affect a GAN's training. The specific design of these formulations is typically guided by the desire to minimize a specific divergence. However, a GAN does not need to decrease a divergence at every step in order to reach the Nash equilibrium [39]. In this situation, an automatic loss-function optimization system may find novel GAN loss functions with more desirable properties. Such a system is presented in Chapter 7 and evaluated on conditional GANs.

2.3 Evolutionary Computation

Since the original theory of natural selection [32, 33], our understanding of the mechanisms of evolution has grown. In the biological realm, evolution has produced a vast array diverse species, proving its efficacy as a framework

for optimization and open-ended discovery. Through the mid-to-late 20th century, with the advent of increasingly sophisticated computers, simulations of evolution were implemented [42]. Methods such as evolutionary strategies [138] and genetic algorithms [76] were developed and found useful in many engineering applications.

In numerous engineering applications, such as antenna design [107], genetic algorithms have been able to find creative solutions that elude humans [100]. Genetic algorithm can represent different search spaces through different representations. This flexibility has made them a technique that is applicable to many problems in unique domains.

Many problems can be framed as numerical optimization problems. While gradient-based methods have been successful at tackling such problems, not all search spaces have known derivatives or are even smooth. To solve theses types of problems, evolutionary strategies implement stochastic, non-convex optimization. In particular, covariance-matrix adaptation evolutionary strategy (CMA-ES) [65] is a widely applicable method. In an iterative manner, candidates are sampled from a multivariate Gaussian distribution, built from a covariance matrix that is updated after each iteration. In this manner, the area in the search space sampled at each step grows, shrinks, and moves dynamically as needed to maximize sampled candidates' fitnesses.

Within computer science itself, evolutionary computation has been used to evolve programs, an idea first theorized by Turing in the mid 20th century [174, 175]. This discipline is known today as genetic programming. Genetic

programming systems are often able to discover solutions to problems that outperform human-built solutions [93].

Early work aimed at developing pattern recognition systems using rule-sets represented as trees [40]. Recently, genetic programming succeeded in discovering nontrivial mathematical formulas that underly physical systems using noisy experimental data [155]. In this system, formulas are represented by directed, acyclic graphs. Remarkably, the system was able to discover Hamiltonian and Lagrangian equations of different mechanical systems, even on a highly-chaotic double-pendulum domain. This work helped inspire the development of GLO, described in Chapter 4.

Approaches to genetic programming that are not based on tree-like structures also exist. Linear genetic programming [11], where programs are represented as a sequence of instructions, and Cartesian genetic programming [116], where programs can be represented by graphs, are two seminal such approaches. Such systems often contain non-coding genes, which have been found to yield more robust evolution [30].

Since evolutionary computation systems are programs themselves, they can be evolved using genetic programming techniques; this idea is known as autoconstructive evolution. A notable example is the Pushpop autoconstructive evolution system [160], where Push programs are evolved alongside the system's evolutionary mechanisms. Push is a general stack-based language with types that was developed specifically for genetic programming.

Evolutionary computation is thus successful at directly evolving complex mathematical solutions to problems, and even at evolving the way in solutions are found. This motivates the use of such techniques to evolve loss functions, as is presented in this dissertation.

2.4 Metalearning

As deep learning systems have become more complex, their architectures and hyperparameters have become increasingly difficult and time-consuming to optimize by hand. In fact, many good designs may be overlooked entirely by humans with prior biases. Metalearning has risen as a field that attempts to tackle this problem from numerous different angles [101], using different approaches that range from Bayesian optimization to evolutionary computation.

Hyperparameter and algorithm selection were identified early on [141] and continue to play an important role today. More advanced early metalearning literature took inspiration from educational psychology, where metalearning is defined as an awareness and subsequent adjustment of an agent's approach to learning [16]. A seminal exploration of metalearning leveraged evolution to attempt to learn how to learn in self-referential, recurrent neural networks [153]; in fact, the author even suggests that meta-metalearning could be advantageous. Subsequent work also attempted to learn how to find learning rules for neural networks, using both gradient descent and genetic algorithms [15]. However, many of these early ideas were infeasible with the computational power that was available at the time.

Neural architecture search has more recently become a key area in the metalearning literature, as the structure of neural networks has been found to be increasingly important to their functionality [38]. Numerous approaches have been proposed, with search strategies drawn from areas such as reinforcement learning, Bayesian optimization, simulated annealing, and evolution (often called neuroevolution in this context). A landmark technique in the literature, NeuroEvolution of Augmenting Topologies (NEAT) [165] has served as the base for more sophisticated and robust architecture search techniques [164]. One of them is CoDeepNEAT [115], which makes use of coevolution to build structure hierarchically. Various neural architecture search techniques have created networks with state-of-the-art performance in different domains [26, 136, 203]. Recently, it was found that that networks that are agnostic to the values of weights can be contructed, exemplifying the importance of structure [44]. Further, even randomly-structured networks with certain properties can learn well [195].

In addition to hyperparameter optimization and neural architecture search, new opportunities for metalearning have recently emerged. In particular, learning rate scheduling can have a significant impact on a model's performance. Such schedules determine how the learning rate changes as training progresses. This functionality is employed by different gradient-descent optimizers, such as AdaGrad [37] and Adam [92]. While monotonically decreasing learning rates generally yield good results, new ideas, such as cyclical learning rates [158], have shown to lead to better models in fewer epochs in some cases.

Metalearning methods have also recently been developed for data augmentation. One such technique is AutoAugment [31], a reinforcement learning based approach to find new data augmentation policies. In reinforcement learning tasks, EC has proven to be a successful approach. For instance, in evolving policy gradients [77], the policy loss is not represented symbolically, but rather as a neural network that convolves over a temporal sequence of context vectors. In reward function search [129], the task is framed as a genetic programming problem, leveraging PushGP [160].

Prior to this research, no existing work in the metalearning literature focused on optimization of loss functions for neural networks. As shown in this dissertation, evolutionary computation can be used in this role to improve neural network performance, to gain a better understanding of the processes behind learning, and to help reach the ultimate goal of fully automated learning.

2.5 Regularization

Regularization has emerged as a central theme in the development of modern neural networks. Traditionally, regularization refers to methods for encouraging smooth mappings by adding a regularizing term to the objective function, i.e., to the loss function in neural networks. Regularization can be defined more broadly, however, as "any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error" [56]. To that end, many regularization techniques have been developed

that aim to improve the training process in neural networks. These techniques can be architectural in nature, such as Dropout [162] and Batch Normalization [82], or they can alter a different aspect of the training process, such as label smoothing [167] or the minimization of a weight norm [67]. These techniques are briefly reviewed in this section, providing context for loss-function metalearning, which provides a new, powerful form of regularization.

2.5.1 Implicit Biases in Optimizers

It may seem surprising that neural networks that are typically highly overparameterized are able to generalize at all. They have the capacity to memorize a training set perfectly, and in fact sometimes do (i.e., zero training error is reached). Different optimizers have different implicit biases that determine which solutions are ultimately found. These biases are helpful in providing implicit regularization to the optimization process [126]. Such regularization results from a network norm—a measure of complexity—that is minimized as optimization progresses. This process is why models continue to improve even after training set has been memorized (i.e., the training error global optima is reached) [128].

For example, the process of stochastic gradient descent (SGD) itself has been found to provide regularization implicitly when learning on data with noisy labels [19]. In overparameterized networks, adaptive optimizers find very different solutions than the basic SGD. These solutions tend to have worse generalization properties, even though they have lower training errors

[191].

2.5.2 Regularization Approaches

While optimizers may minimize a network norm implicitly, regularization approaches supplement this process and make it explicit. For example, a common way to restrict the parameter norm explicitly is through weight decay. This approach discourages network complexity by placing a cost on weights [67].

Generalization and regularization are often characterized at the end of training, i.e. as a behavior that results from the optimization process. Several findings have influenced work in regularization. For example, flat loss landscapes have better generalization properties [24, 90, 102]. In overparameterized cases, the solutions at the center of these landscapes may have zero training error (i.e., perfect memorization), and under certain conditions, zero training error empirically leads to lower generalization error [14, 122]. However, in most cases when a training loss of zero is reached, generalization suffers [83]. This behavior can be thought of as overfitting, and techniques have been developed to reduce it at the end of the training process, such as early stopping [119] and flooding [83].

Both early stopping and flooding assume that overfitting happens at the end of training, which is not always true [48]. In fact, the order in which easy-to-generalize and hard-to-generalize concepts are learned is important the network's ultimate generalization. For instance, larger learning rates early in

the training process often lead to better generalization in the final model [104]. Similarly, low-error solutions found by SGD in a relatively quick manner—such as through high learning rates—often generalize better [196].

Other techniques tackle overfitting by making it more difficult. Dropout [162] makes some connections disappear. Cutout [35], Mixup [199], and their composition, CutMix [197], augment training data with a broader variation of examples.

Notably, regularization is not a one-dimensional continuum. Different techniques regularize in different ways that often interact. For example, flooding sometimes cancels out generalization from early stopping [83]. However, ultimately all regularization techniques alter the gradients that result from the training loss. This observation suggests that loss-function optimization might be an effective way to regularize the training process.

2.5.3 Auxiliary Classifiers

Auxiliary classifiers are small sub-networks within a deep model that predict outputs given internal activations from the parent model. During training, losses are calculated for the main network and are summed with scaled losses from one or more auxiliary classifiers. Training gradients propagate through the network in an end-to-end manner, and auxiliary classifiers may be discarded at inference-time.

Auxiliary classifiers were originally developed for the GoogLeNet architecture [166], where they were intended to help the network learn lower-level

features, facilitate the propagation of gradients to early stages of the network, and provide regularization. GoogLeNet used two auxiliary classifiers, resulting in the end-to-end loss of $\mathcal{L}_{\text{Final}} = \mathcal{L}_{\text{Main}} + 0.3\ \mathcal{L}_{\text{Aux1}} + 0.3\ \mathcal{L}_{\text{Aux2}}$.

In the subsequent Inception-v3 architecture [167], an extension of the original GoogLeNet, a single auxiliary classifier was used. The removal of the early-stage auxiliary classifier did not degrade performance. Additionally, there were no noticable benefits to auxiliary classifiers in early phases of training. These two observations prompted the authors to argue that regularization was the primary benefit confered by auxiliary classifiers, rather than a way to learn lower-level features more effectively.

2.6 Conclusion

The literature review in this chapter showed how design decisions can impact on the performance of machine learning models. Designs are often very complex and have non-intuitive interactions between components. Metalearning has tackled this from many angles, using technologies such as evolutionary computation to automatically optimize different design elements. However, metalearning has not yet been used to optimize loss functions. These functions represent the primary training objective in neural networks and thus present a unique opportunity for metalearning. By directly impacting the optimization process, loss-function metalearning can alter training dynamics in useful ways.

Chapter 3

Experimental Methodology

Deep learning is a very detail-sensitive discipline; the way that experiments are conducted can lead to a different interpretation of subsequent results. This chapter delves into the various aspects of the experimental setup and methodologies that are used in this dissertation's embodied work. First, the image classification benchmark datasets that were used to evaluate GLO and TaylorGLO are described, followed by an overview of the model architectures that were trained on these datasets. Lossferatu and Fumanchu, two components of a novel distributed experiment management system are then described. They provide experiment host and model training and evaluation functionality, respectively. Finally, the statistical testing methodology used to evaluate the significance of results is justified and compared with alternatives.

3.1 Datasets

The empirical analyses in this dissertation make use of standard benchmark datasets. The MNIST, CIFAR-10, CIFAR-100, and SVHN datasets, and their training configurations, including data splits and processing and augmentation pipelines, are described below.

3.1.1 MNIST

The first domain used for evaluation was MNIST Handwritten Digits [99], a widely used dataset where the goal is to classify 28×28 pixel images as one of ten numerical digits. Each image has a single channel. MNIST is composed of 55,000 training samples, 5,000 validation samples, and 10,000 testing samples. The dataset is well understood and relatively quick to train.

3.1.2 CIFAR-10 and CIFAR-100

To validate GLO and TaylorGLO in a more challenging context, the CIFAR-10 [94] dataset was used. It consists of small 32×32 pixel color photographs of objects in ten classes. CIFAR-10 traditionally consists of 50,000 training samples, and 10,000 testing samples; however 5,000 samples from the training dataset were used for validation of candidates, resulting in 45,000 training samples.

Models were trained with their respective hyperparameters from the literature. Inputs were normalized by subtracting their mean pixel value and dividing by their pixel standard deviation. Standard data augmentation techniques consisting of random horizontal flips and croppings with two pixel padding were applied during training.

CIFAR-100 is a similar, though significantly more challenging, dataset where a different set of 60,000 images is divided into 100 classes, instead of 10. The same splits for training, validation, and testing were used for CIFAR-100 as for CIFAR-10, and evaluate TaylorGLO further.

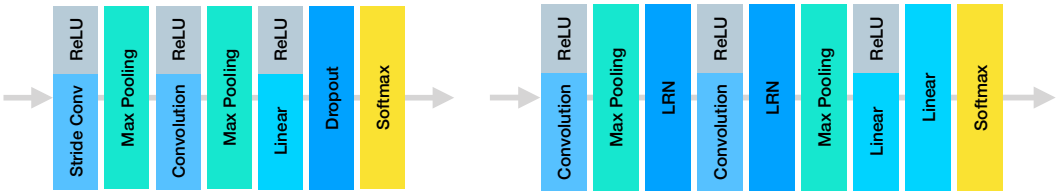
3.1.3 Street View House Numbers (SVHN)

The Street View House Numbers (SVHN) [124] dataset is another image classification, consisting of 32×32 pixel images of numerical digits from Google Street View. It was used to further evaluate TaylorGLO in this dissertation. SVHN consists of 73,257 training samples, 26,032 testing samples, and 531,131 supplementary, easier training samples. To reduce computation costs, supplementary examples were not used during training, which explains why presented baselines are lower in the experiments contained in this dissertation than other SVHN baselines in the literature. Since a validation set is not in the standard splits, 26,032 samples from the training dataset were used for validation of candidates, resulting in 47,225 training samples.

As with CIFAR-10 and CIFAR-100, models were trained with their respective hyperparameters from the literature and with the same data augmentation pipeline.

3.2 Evaluated Model Architectures

Neural networks come in many different morphologies, with varying structural motifs, sizes, and design principles. The experiments in this dissertation aim to cover a variety of architectures, including those with state-of-the-art results. Notably, many of these architectures have been extensively tuned to work well with the cross-entropy loss, providing a high-bar for metalearned loss functions. Specific architectures, and some of their noteworthy properties, are described below. These architectures have all been designed



(*a*) MNIST (*b*) CIFAR-10

Figure 3.1: **Basic CNN Architectures.** Both architectures are standard, relatively shallow CNNs with sequential layers. Samples flow from left to right. These architectures were used to evaluate the GLO technique.

to perform well when trained with the cross-entropy loss function. Following these descriptions is a quantitative comparison of the architectures' relative sizes and training times.

3.2.1 Basic CNNs

Two simple, relatively shallow convolutional neural network (CNN) architectures were used for initial analyses of the GLO technique, one each for the MNIST and CIFAR-10 datasets (Figure 3.1).

MNIST A simple CNN architecture with the following layers was used: (1) 5×5 convolution with 32 filters and ReLU [121] activations, (2) 2×2 stride-2 max-pooling, (3) 5×5 convolution with 64 filters and ReLU activations, (4) 2×2 stride-2 max-pooling, (5) 1024-unit fully-connected layer with ReLU activations, (6) a dropout layer [75] with 40% dropout probability, and (7) a softmax layer. Training uses stochastic gradient descent (SGD) with a batch size of 100, a learning rate of 0.01, and, unless otherwise specified, for 20,000 steps.

CIFAR-10 A simple CNN architecture, taken from [51] (and itself inspired by AlexNet [95], which is described in Section 3.2.2), with the following layers was used: (1) 5×5 convolution with 64 filters and ReLU activations, (2) 3×3 max-pooling with a stride of 2, (3) local response normalization [95] with $k = 1, \alpha = 0.001/9, \beta = 0.75$, (4) 5×5 convolution with 64 filters and ReLU activations, (5) local response normalization with $k = 1, \alpha = 0.001/9, \beta = 0.75$, (6) 3×3 max-pooling with a stride of 2, (7) 384-unit fully-connected layer with ReLU activations, (8) 192-unit fully-connected, linear layer, and (9) a softmax layer. This architecture contains components that are typical in modern neural network architectures. As a result, this simple architecture slightly outperforms AlexNet while being shallower.

Inputs to the network are sized $24 \times 24 \times 3$, rather than $32 \times 32 \times 3$ as provided in the dataset; this smaller input size enables more sophisticated data augmentation. The data augmentation steps consist of: random 24×24 croppings selected from each full-size image—to force the network to learn spatial invariance better—that are randomly flipped longitudinally, randomly lightened or darkened, and their contrast randomly perturbed. Furthermore, to attain quicker convergence, an image's mean pixel value and variance were subtracted and divided, respectively, from the whole image during training and evaluation. CIFAR-10 networks were trained with SGD, L_2 regularization with a weight decay of 0.004, a batch size of 1024, and an initial learning rate of 0.05 that decays by a factor of 0.1 every 350 epochs.

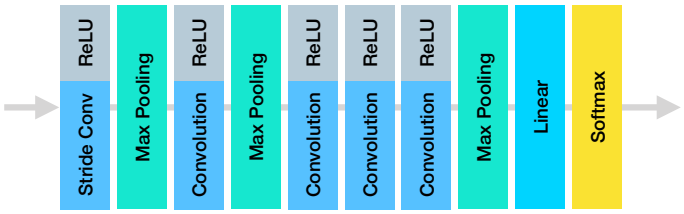


Figure 3.2: **AlexNet Model Architecture.** AlexNet was a seminal CNN architecture. Samples flow through the network, as tensors are gradually down-sampled, to arrive a final set of scaled classification logits. AlexNet provides a fixed, basic CNN architecture that is used to evaluate TaylorGLO.

3.2.2 AlexNet

AlexNet [95] is a relatively early CNN with state-of-the-art results on CIFAR-10 at its time. AlexNet is composed of a sequence of convolution, ReLU, and max pooling layers, followed by a final linear classification layer (reference Figure 3.2).

AlexNet is a seminal architecture with a conventional sequential design. Additionally, its short training times make it a compelling architecture to iterate on. This fixed architecture is used to evaluate TaylorGLO.

3.2.3 AllCNN

The All-Convolutional Neural Network (AllCNN) [161] is a unique type of architecture in that it is entirely composed of convolutions, strided convolutions, dropout layers, and one single average pooling layer at the end of the network (reference Figure 3.3). Components that CNNs traditionally contain, such as spatial pooling, fully-connected layers, and batch normalization [82], are entirely absent from AllCNN models.

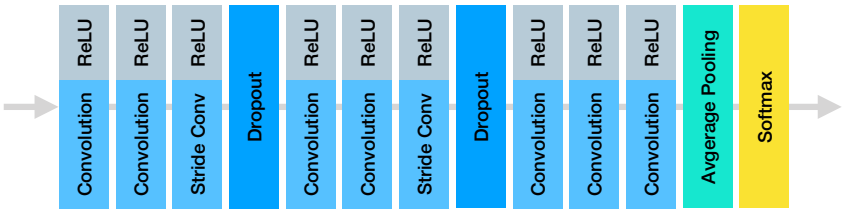


Figure 3.3: **AllCNN-C Model Architecture.** Unlike many other architectures, AllCNNs only use convolution and dropout regularization layers, along with a final average pooling layer in lieu of a fully connected classifier. AllCNN-C provides a unique architecture that is used to evaluate TaylorGLO.

Specifically, the AllCNN-C variant of the AllCNN architecture is used to evaluate TaylorGLO. AllCNNs provide a unique architecture that achieves high accuracy while forgoing the use of many typical network features.

3.2.4 ResNet

Residual Networks (ResNets) [71] were a novel network morphology that allows extremely deep networks with up to hundreds of layers to be trainable. It contains residual connections (a type of skip connection) that alleviate the vanishing gradients problem (reference Figure 3.4). Residual blocks—groups of layers with a residual skip connection—are arranged sequentially many times within each of three modules. Downsampling occurs between modules (downsampling operations are omitted from Figure 3.4 for brevity). ResNets are configured by a depth parameter from which the whole architecture is built.

ResNets were used to evaluate TaylorGLO since they are a seminal network that introduced architectural motifs that are common to many modern

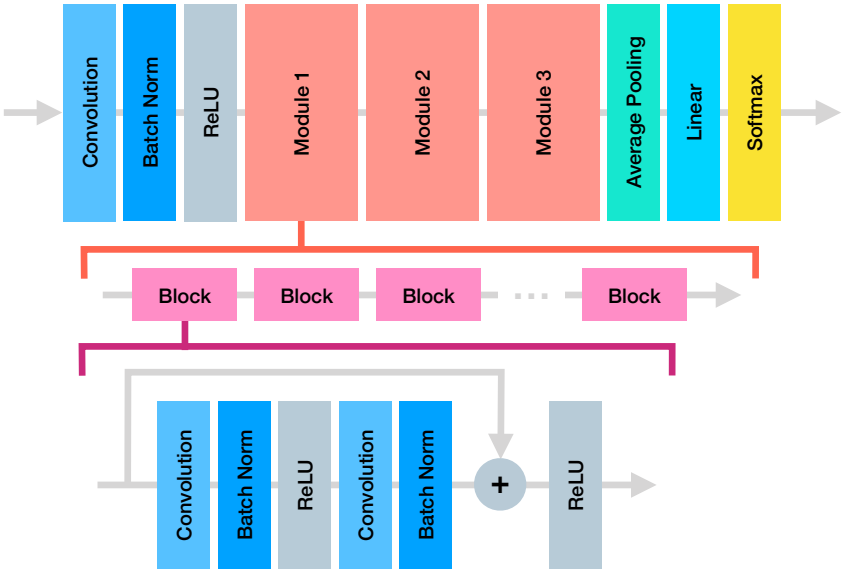


Figure 3.4: **ResNet Model Architecture.** Modules composed of sequences of residual blocks make up the main structure of ResNet models. Residual blocks contain skip connections that provide a more direct path for gradients to propagate, allowing deeper networks to be trained.

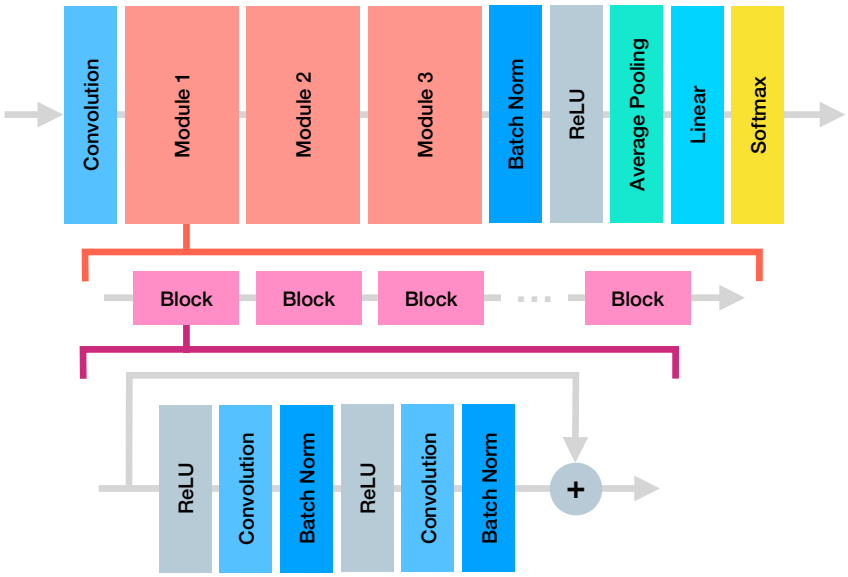


Figure 3.5: **Preactivation ResNet Model Architecture.** A small modification to traditional ResNets, Preactivation ResNets are able to learn better by providing a clearer path for gradients to propagate. This natural extension to the base ResNet architecture is used to evaluate TaylorGLO.

neural networks.

3.2.5 Preactivation ResNet

Preactivation ResNets [72] emerged from an analysis of signal propagation in traditional ResNets. By simply changing the order in which ReLU units are applied, Preactivation ResNets are able to achieve higher accuracy and better generalization properties. As seen in Figure 3.5, there is a clearer path for gradients to propagate, compared to traditional ResNets (reference Figure 3.4), due to the lack of a ReLU layer between blocks.

Preactivation ResNets, in conjunction with ResNets, serve as a good

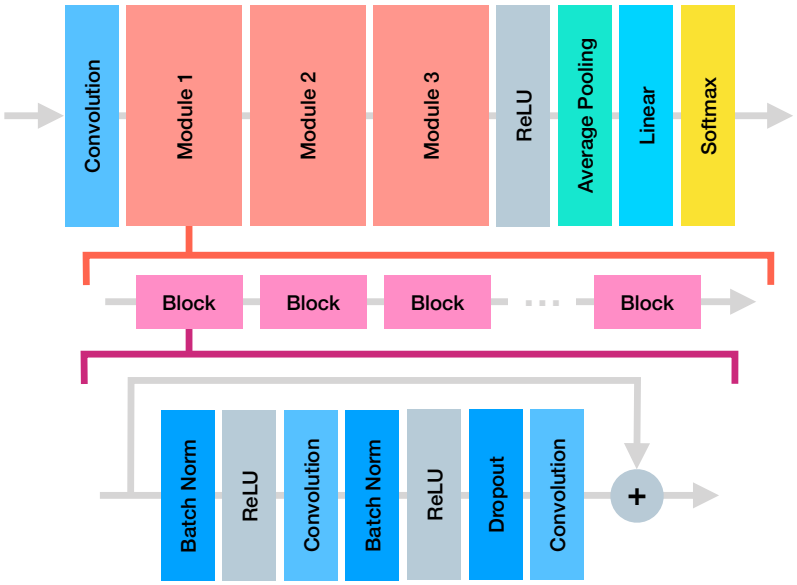


Figure 3.6: **Wide ResNet Model Architecture.** Wide ResNets provide a significant departure from the "deep and skinny" configuration that is typical to ResNets. Wide ResNets are a good fit evaluate TaylorGLO on wider networks.

way to evaluate how TaylorGLO performs on two similar, but slightly different, types of networks with different gradient propagation characteristics.

3.2.6 Wide ResNet

Wide ResNets [198] tackle the marginal returns gained in performance and large increases in training time associated with increasingly deep ResNets. By utilizing a wider network with small modifications (reference Figure 3.6), Wide ResNets are able to outperfom their narrow counterparts, while requiring over an order of magnitude fewer layers. Wide ResNets have configurable depth and width parameters. Notably, Wide ResNets include dropout layers between

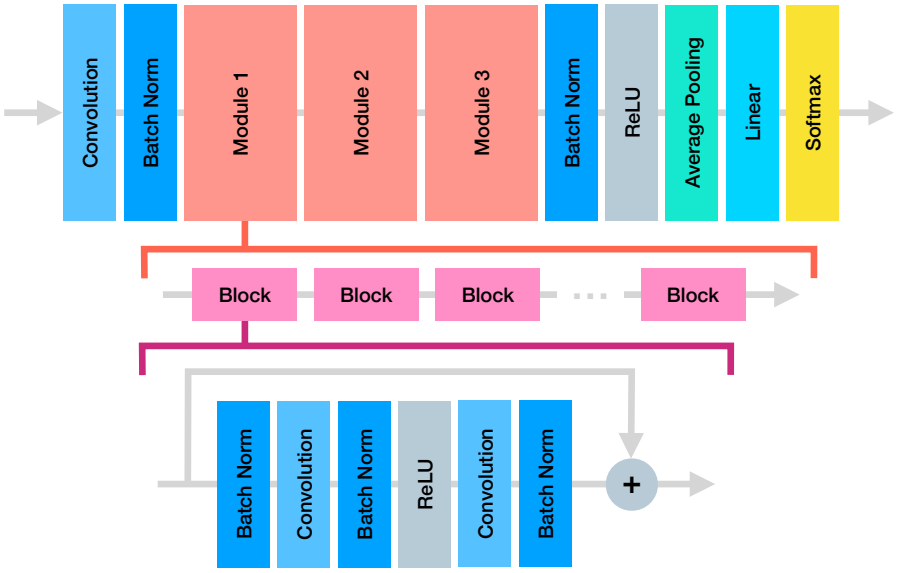


Figure 3.7: **PyramidNet Model Architecture**. PyramidNets are a more recent member of the ResNet lineage that has tuned various aspects of the network to perform better. The PyramidNet architecture provides a way to evaluate TaylorGLO on a recent, highly tuned architecture.

convolutions, a feature that improves performance on Wide ResNets but had been found to be detrimental on previous ResNet architectures [72].

Wide ResNets are widely used in the literature, making them a must-have architecture in any well-rounded evaluation. While the general flow of gradients through Wide ResNets is very similar to other residual networks, their width and depth can vary significantly, providing a direct point of comparison for the effects of width and depth on TaylorGLO.

3.2.7 PyramidNet

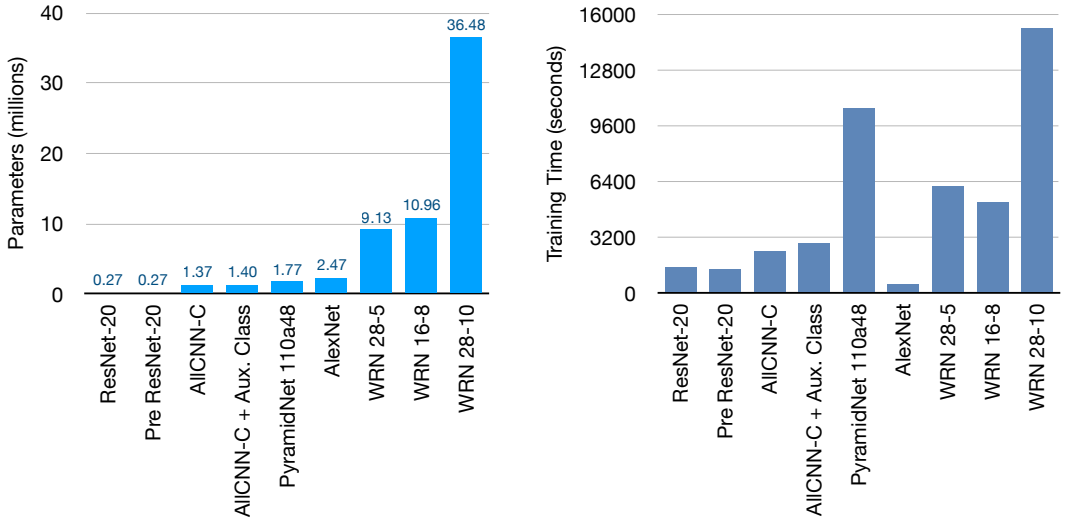
Deep Pyramidal Residual Networks (PyramidNets) [63] are yet another network with residual connections whereby channel dimensionality gradually increases—as opposed to staying static—in between downsampling opperations. These increases may happen in either an additive or multiplicative manner.

As another mature, modern ResNet variant, PyramidNets are used in the TaylorGLO evaluation, namely on the CIFAR-100 dataset. PyramidNets are an example of an architecture that has been extensively tuned for high performance when training with the cross-entropy loss, providing a high bar for TaylorGLO to exceed.

3.2.8 Architecture Comparisons

Overall, these architectures provide a test suite upon which the techniques presented in this dissertation can be evaluated. The architectures described above are varied in their components, approaches, and sizes, thus providing a representative sampling of modern deep neural networks.

Figure 3.8 provides a comparison of the number of trainable parameters and training times across the evaluated architectures. The parameter counts range across three orders of magnitude. Training times range from under ten minutes for AlexNet, to over four hours for Wide ResNet 28-10. These values exemplify one dimension of the variety in the evaluated architectures.



(a) Trainable Parameters (b) Training Times

Figure 3.8: **Comparing architecture trainable parameters** (a) **and training times** (b)**.** The number of trainable paramters in evaluated architectures span multiple orders of magnitude. While there tends to be a correlation between the number of parameters and training time, certain architectures have disproportionately higher (e.g., PyramidNet) or lower (e.g., AlexNet) training times.

Training times are calculated by multiplying a per-batch training time measurement by the number of batches per epoch and the number of epochs per model. To ensure stability, these per-batch training time values are the average batch training time from the second epoch of training (i.e., a warm start). All values were calculated from training on a machine with a 10-core Intel Xeon Gold 5215 processor running at a base frequency of 2.5GHz, NVIDIA GeForce RTX 2080 Ti GPU, 96GB of memory, and a solid-state drive connected through NVMe. No other work was running on the machine while each profiling experiment took place.

3.3 System Architecture

Loss function evolution typically requires hundreds of neural networks to be evaluated. Thus, these evaluations happen in parallel for efficiency, requiring a distributed system for running experiments on a cluster of machines with dozens of GPUs. A custom system was built to such an end for this body of work. This system was created since no comprehensive, extensible, and usable system for managing and running evolution experiments with distributed candidate evaluations was found to exist. This dissertation presented an opportunity to develop such a system.

3.3.1 Overview

The system is composed of two key components that interact with each other:

Lossferatu: the parallelized experiment host that runs evolutionary processes, manages results, and coordinates candidate evaluation. Lossferatu can run for extended periods of time without human intervention, much like its namesake Nosferatu [3].

Fumanchu: a generic, model-agnostic neural network training and evaluation component with a unified interface. One experiment may involve hundreds of unique invocations of Fumanchu. More informally, Fumanchu treats models as cattle, rather than as pets; the inspiration for being named after Fu Manchu the bull [113].

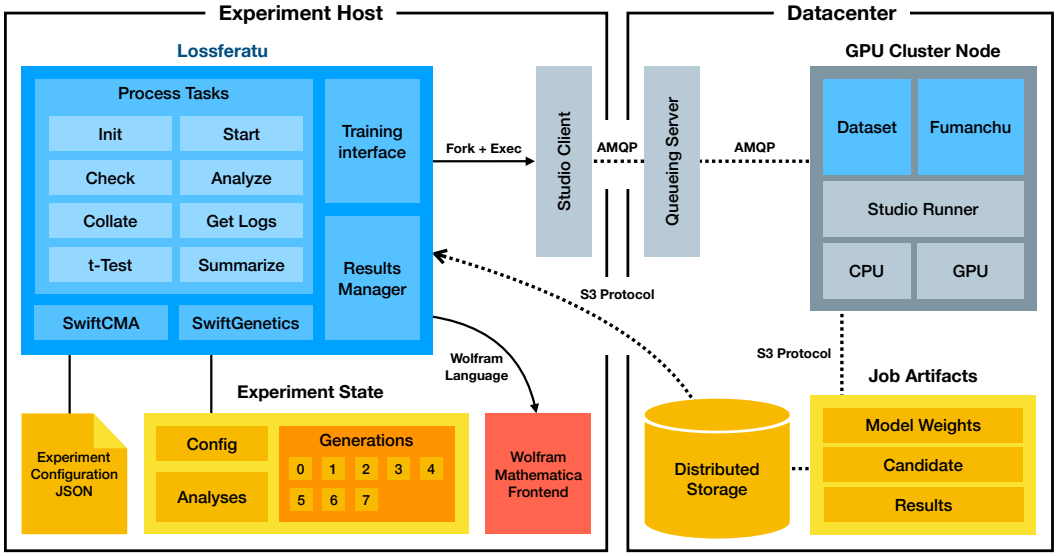


Figure 3.9: **System architecture for distributed experiment execution.** A single experiment host, running Lossferatu, serves as the central coordination point for experiments. Individual models are dispatched for training and evaluation on a cluster of machines running Fumanchu. In this manner, evolution experiments with long-running evaluations can be configured and efficiently run.

Lossferatu uses the Studio [178] model management framework to dispatch atomic fragments of work, i.e. jobs. Each job corresponds to a single candidate evaluation and consists of a run of Fumanchu. Studio jobs are referenced in a RabbitMQ [177] message queue that conforms to the Advanced Message Queuing Protocol (AMQP) [84]. Job code, datasets, results, and job artifacts are stored in MinIO [2], a distributed object store that is accessible with the AWS S3 API [4]. Jobs are consumed from the queue by Studio Go Runners [120] as resources become available. Studio Go Runners are decentralized entities that manage the execution of jobs and resource allocation.

Studio Go Runners run as kubelets within a Kubernetes [1] cluster. Concurrently, Lossferatu monitors for completed jobs and processes their results as needed.

The general architecture of this system is depicted in Figure 3.9. Both Lossferatu and Fumanchu, their capabilities, and their interactions with the components in the figure, are described in more detail in the following subsections.

3.3.2 Lossferatu

Lossferatu is a multithreaded program that runs on the host machine and provides a centralized user interface and core functionality for experiments. It was designed with the following desiderata in mind:

Safety and Continuity: Experiments should be able to handle malfunctions gracefully throughout the software stack, and surface them to the user if recovery is not possible. Additionally, experiments should be able to be stopped, continued, or restarted at any point and must even be able to gracefully handle manual state changes.

Extensibility: Lossferatu should support new types of experiments in a backwards-compatible manner without significant changes.

Full State Persistence: During iterative experimentation or analysis, certain components of the experiment's state may be useful to know. Which

pieces of state are useful may not be known *a priori*, so all state should be persisted in human-interpretable JSON files.

Unified Experiment Structure: All experiments should be stored in a consistent manner on the file system, and have consistently formatted experiment configuration files.

Simple User Interface: An easy-to-use command-line interface for core functionality should be exposed, with more sophisticated behaviors abstracted into shell scripts.

Automated Analysis Pipelines: Results should be easily processed and collated automatically. Evolution experiments should be able to generate Mathematica [140] notebooks that summarize results and provide publication-quality plots.

Lossferatu was designed such that atomic steps in an experiment each correspond to a single, multi-threaded process. Such a philosophy yields a system that is amenable to manual intervention and is naturally safe and efficient. Consequently, a single evolution experiment may result in thousands of small, fast-completing processes running over the course experiment. This design also requires state to be maintained outside the confines of a process; the file system is used for such an end.

Multigeneration Lossferatu experiments have a directory structure as shown in Figure 3.10. Lossferatu automatically creates and manages this struc-

ture from a given experiment configuration JSON file. Generations are segregated into numbered directories where their respective state is maintained. Since Lossferatu processes are stateless, these generation directories can be manually altered in incomplete experiments.

For each experiment, Lossferatu can perform a holistic analysis: calculating per-generation metrics, creating experiment configuration files for final evaluation of the best candidate at each generation, and creating a Mathematica notebook that summarizes experiment results. Lossferatu creates the Mathematica notebook without user involvement by generating Wolfram code for the notebook and running it through the Mathematica front-end in the background using the `wolframscript` tool. Several of the plots in this dissertation were directly exported from these generated notebooks. The generated configuration files for each generation can then be used by the autogenerated `experiment_best_candidate_one_shot` script to run child experiments, typically for evaluating best candidates on a testing dataset. This capability greatly reduces the amount of effort involved in evaluating specific candidates.

Generational (i.e., evolution) experiments are conveniently run through repeated calls of the `run_generation` script. Since evaluations happen asynchronously, on remote servers, each generation is split into two phases: a start phase, where new candidates are generated and submitted to the queue for evaluation, and a check phase, where MinIO is periodically polled to check experiment progress and fetch and process results if needed.

However, not all experiments fall into a generational modality; such

as baseline experiments and secondary candidate evaluations. Lossferatu includes support for these "one-shot" jobs as well. One-shot jobs share many commonalities with generational jobs, such as the two-phase experiment execution system (i.e., submit a job and poll for results).

The generated Lossferatu scripts are simple contrivances that explicitly invoke Lossferatu with various parameters. Lossferatu's command-line interface is structured around the following commands:

init: Initializes a new experiment directory given an experiment configuration JSON file.

start: Creates a new generation directory, creates new candidates, and submits them to the queue for evaluation. Queue submissions can be configured to happen either sequentially or concurrently.

check: Polls MinIO for updated job output files. If any jobs have finished running, their results are downloaded, processed, and persisted. The command returns a unique exit code once all jobs have finished running.

analyze: Analyzes completed generations and creates a Mathematica notebook with statistics, plots, and other details.

getinvocation: Prints the exact Fumanchu invocation that would be run for a given one-shot experiment configuration file. This command makes it effortless to rerun exact one-shot experiments on any machine that has

Fumanchu and is not a part of the cluster to which Lossferatu dispatches work.

studiolog: Downloads and prints the Studio output log for a given job identifier. Such a command is useful for manually monitoring the progress of in-flight jobs, and seeing raw results.

ttest: Given two directories with equal numbers of one-shot jobs, various statistical significance tests are run on their primary evaluation metrics as defined in their experiment configurations.

collateoneshots: Takes a directory of one-shot experiments and collates their results into a CSV file at the directory's root.

resummarize: Creates a results summary CSV for a given directory of directories of one-shot experiments. In practice, this command allows for baselines to be easily summarized.

resummarizegenerational: Creates a results summary CSV for a given directory of generational experiments. In practice, this command is used to summarize the results for the best candidates from a set of evolution experiments and cross-reference them with respective one-shot baselines.

test: Runs various unit-tests of Lossferatu.

Lossferatu is primarily written in the Swift programming language, leveraging the SwiftCMA [49] package for its CMA-ES implementation and the SwiftGenetics [50] package for building genetic algorithms. Multi-core concurrency in

Lossferatu is implemented atop `libdispatch` [81]. Lossferatu is able to run on both macOS and Linux, and, notably, multiple experiment hosts running their own instances of Lossferatu can be used concurrently for different experiments on the same cluster.

Lossferatu is arguably the first comprehensive system for managing, running, and analyzing evolution experiments with distributed candidate evaluations. Lossferatu provides extensible functionality and a user-friendly interface that enables new experiments to be easily configured.

3.3.3 Fumanchu

Fumanchu is a highly configurable model training and evaluation framework. Lossferatu runs Fumanchu to evaluate individual candidates. It was designed to achieve the following goals:

Model and Dataset Agnosticism: Large components of Fumanchu should function generically, regardless of the model or dataset that is configured. Moreover, this property should hold true across different types of tasks (e.g., classification, regression) when possible. This property makes adding new model architectures and domains an easier, less error-prone endeavor.

Command-Line Interface: Runs of Fumanchu should be configured entirely through a command-line interface. This design decision facilitates interaction with Fumanchu and simplifies manual invocation by users.

Logging, Pretrained Models, and Analyses: Fumanchu should be able to log relevant metrics and data during the training process, perform various analyses (e.g., loss surface analysis, adversarial attacks) and support the use of pretrained models for training, inference, and analysis.

Fumanchu is resident on the same machine as Lossferatu. Whenever Lossferatu dispatches a unit of work, Fumanchu's code is duplicated to a temporary directory and an invocation shell script with job-specific parameters is created in this directory. Studio then encodes this directory into a tar file, uploads it to MinIO, and appends a new job to the queue. This per-job copying of Fumanchu ensures that no node in the cluster will ever have a stale version of Fumanchu.

Whenever enough resources are available (i.e., half a GPU) on a cluster machine, a Studio Go Runner will consume a job from the queue and start running it once its environment and dependencies are setup. In a startup script that Lossferatu packages into each job, the correct training dataset and, optionally, a pretrained model are copied from MinIO. Fumanchu is then invoked and sequentially performs model training, evaluation, and analysis. The startup script may invoke Fumanchu more than once, as specified in the current Lossferatu experiment configuration, if trained model performance does not exceed a threshold.

Upon completion of its workload, Fumanchu saves trained model artifacts, and large results files from analyses, to MinIO through Studio's model

persistance mechanism. Results are written to a file descriptor that Studio automatically packages into a per-job tar file that is persisted on MinIO. Fumanchu is written entirely in Python and leverages the PyTorch library [131] for training neural networks.

Fumanchu is an important component that Lossferatu leverages for candidate evaluation and analysis. Its independent nature allows it to be manually invoked and used on any machine, outside of Lossferatu experiment management system. Evolution experiments in Lossferatu whose candidate evaluations do not necessarily involve training a neural network could use a different, application-specific component in lieu of Fumanchu.

3.3.4 Hardware Specifications

Lossferatu ran on both (1) a 2016 MacBook Pro notebook computer with a quad-core 2.9GHz Intel Core i7 processor and 16GB of memory and (2) a 2019 MacBook Pro notebook computer with a hexa-core 2.6 GHz Intel Core i7 processor and 16GB of memory. Both of these experiment hosts communicated with the same cluster where candidate evaluations were run.

The cluster was composed of eight machines, with four in each of two separate configurations:

biggpu: Servers with two 16-core Intel Xeon E5-2683 v4 processors running at a base frequency of 2.1GHz, 264GB of memory, and eight NVIDIA GeForce RTX 1080 Ti GPUs.

biggergpu: Servers with two 16-core Intel Xeon Silver 4216 processors running at a base frequency of 2.1GHz, 384GB of memory, and nine NVIDIA GeForce RTX 2080 Ti GPUs.

3.4 Significance Testing

While statistical significance testing is unfortunately not a widespread, standard practice in machine learning literature, it is an important part of any rigorous data analysis. In this body of work, such tests are used frequently to compare resultant performance metrics, such as testing accuracy, from models trained with different techniques. Significance testing can assure that a given technique actually is an improvement over others.

Significance tests define a null hypothesis and reject it if a p-value is below a predefined significance level, typically 0.05. A p-value is the probability of obtaining extreme results at the same level or greater than the results observed given that the null hypothesis is true.

When comparing results in this dissertation, a one-tailed null hypothesis is typically used:

$$H_0 : \neg \left(\mu_1 < \mu_2 \right), \tag{3.1}$$

where μ_1 and μ_2 are mean values from two separate experiments. The rejection of this null hypothesis implies that μ_2 is statistically significantly larger than μ_1. A two-tailed null hypothesis—to test for the dissimilarity of two

distributions—can be similarly constructed as

$$H_0 : \mu_1 = \mu_2. \tag{3.2}$$

The two main options that are available for performing significance testing in this type of setting are Student's t-Test [57] and Welch's t-Test [189]. Both of these tests assume that both groups of data are sampled from populations that are normally distributed. In practice, this assumption holds for trained network accuracies. Figure 3.11 demonstrates this property with 100 AllCNN-C networks trained on CIFAR-10. Student's t-Test asserts additional requirements, where both populations need to have equal numbers of samples and equal variances (in contrast, Welch's t-Test does not have these requirements).

While certain sets of samples on which statistical tests are performed may have equal variances, this is not necessarily known *a priori*. Checking variances in order to decide between the usage of Student's or Welch's t-Test is not recommended practice [202]. Additionally, even in cases where variances and sample counts are known to be similar, it is reasonble to use Welch's t-Test since it is more robust and still has high-statistical power [146]. Thus Welch's t-Test is used exclusively in this dissertation when comparing sets of performanc metrics from different techniques.

Welch's t-Test functions by calculating a t-statistic and degrees of freedom, ν, which are input into the cumulative distribution function (CDF) of the t-distribution to get a level of significance (i.e., a p-value) that signifies the

probability that the null hypothesis is true by random chance. The t-statistic is calculated as follows:

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}} , \tag{3.3}$$

where $\bar{X}_j$ is the sample mean, s_j^2 is an unbiased variance estimator (i.e., sample variance with Bessel's correction), and N_j is the number of samples for the jth population of samples. The dataset's degrees of freedom are approximately calculated using the Welch-Satterthwaite equation [151]:

$$\nu \approx \frac{\left(\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}\right)^2}{\frac{s_1^4}{N_1^2 \nu_1} + \frac{s_2^4}{N_2^2 \nu_2}} , \tag{3.4}$$

where $\nu_j = N_j - 1$. The cumulative probability of the t-distribution for a given t-statistic value and ν is:

$$\text{CDF}(f_\nu(t)) = 1 - \frac{1}{2} I_{\frac{\nu}{t^2+\nu}} \left(\frac{\nu}{2}, \frac{1}{2}\right) , \tag{3.5}$$

where $I_x(a, b)$ is the regularized incomplete beta function. Such a p-value is reported without modification for one-tailed t-tests, while it is multiplied by two for two-tailed t-tests.

Throughout this dissertation, Welch's t-Test is used to determine statistical significance when comparing sets of results. It is a better fit than Student's t-Test due to its higher robustness and statistical power. Graphical figures indicate statsitical significance with standard notation: "ns" indicates $p > 0.05$, "$*$" indicates $p \leq 0.05$, "$**$" indicates $p \leq 0.01$, and "$***$" indicates $p \leq 0.001$.

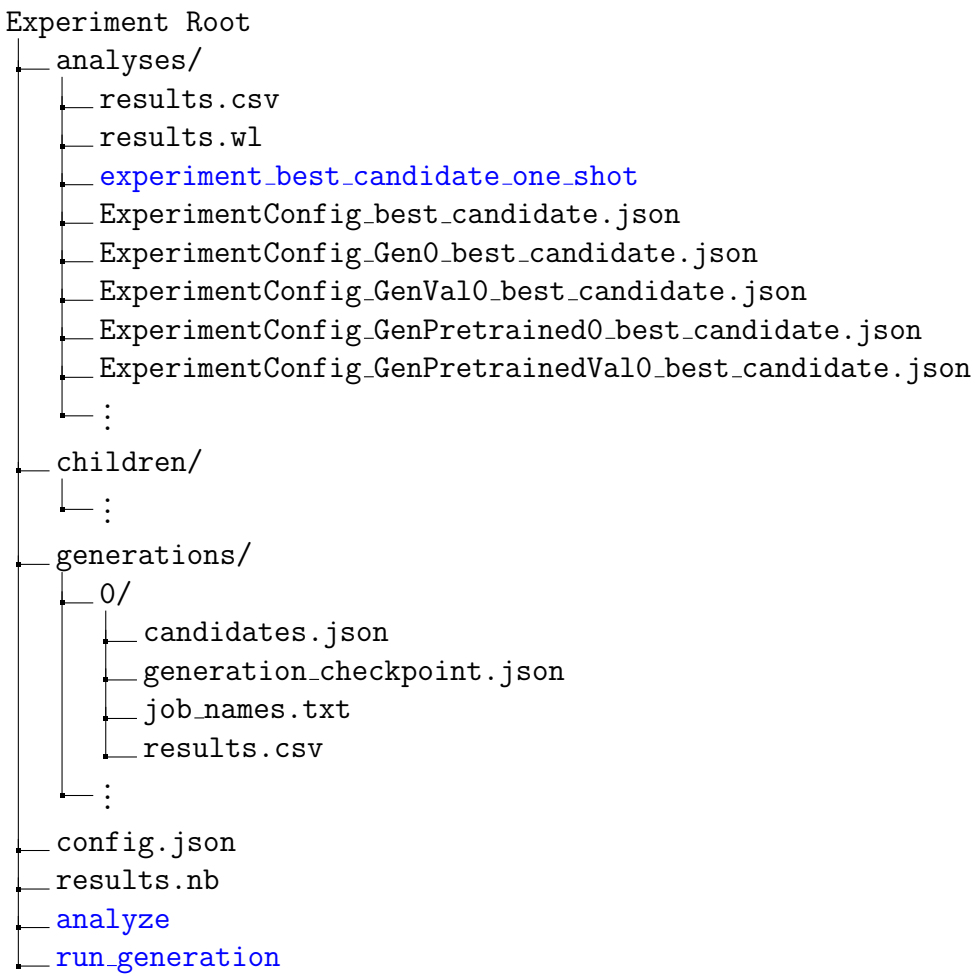


Figure 3.10: **Representative Lossferatu generational experiment directory structure.** Lossferatu stores all state and command scripts (shown in blue) in an intuitive directory structure on the file system. This structure is initialized and managed by Lossferatu, while making manual user intervention possible. An experiment directory contains all internal state, evaluated candidate results, analyses, and child experiments.

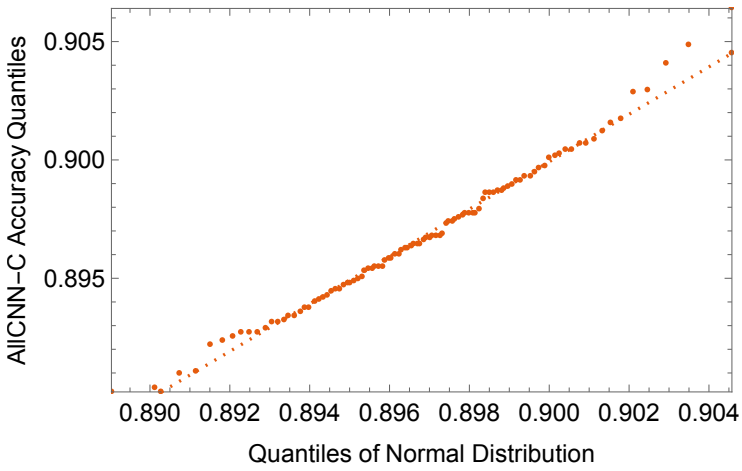


Figure 3.11: **Quantile-quantile plot of 100 trained AllCNN-C networks' testing accuracies against a normal distribution.** The highly-linear distribution of points demonstrates that testing accuracies are normally distributed.

Chapter 4

Genetic Loss-function Optimization (GLO)

In this chapter[1], a general framework for loss function metalearning, covering both novel loss function discovery and optimization, is developed and evaluated experimentally. This framework, Genetic Loss-function Optimization (GLO), leverages genetic programming to build loss functions by representing them as trees, and subsequently a Covariance-Matrix Adaptation Evolution Strategy (CMA-ES) to optimize their coefficients. On the MNIST image classification benchmark, GLO discovered a surprising new loss function, named Baikal for its shape, that outperforms the standard cross-entropy loss in terms of training speed, final accuracy, and data requirements.GLO's highly flexible representation also provides ample opportunities for extensibility; one such extension is explored in this chapter.

4.1 Method

The task of finding and optimizing loss functions can be framed as a functional regression problem. GLO accomplishes this task through two

[1]The work in this chapter was previously presented at the 2020 IEEE Congress on Evolutionary Computation (CEC) [52]. Risto Miikkulainen provided guidance and feedback through discussions.

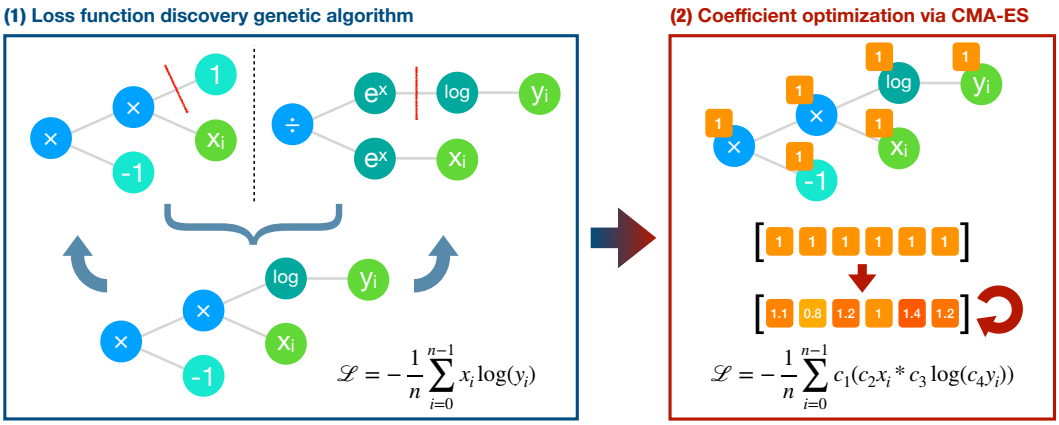


Figure 4.1: **Genetic Loss Optimization (GLO) overview.** A genetic algorithm constructs candidate loss functions as trees. The best loss function from this set then has its coefficients optimized using CMA-ES. GLO loss functions are able to train models more quickly and more accurately than the cross-entropy loss.

high-level steps (shown in Figure 4.1): **(1) loss function discovery:** using approaches from genetic programming, a genetic algorithm builds new candidate loss functions, and **(2) coefficient optimization:** to further optimize a specific loss function, a covariance-matrix adaptation evolutionary strategy (CMA-ES) is leveraged to optimize coefficients. Each of these steps is described in detail below.

4.1.1 Loss Function Discovery

GLO uses a population-based search approach, inspired by genetic programming, to discover new optimized loss-function candidates. Under this framework, loss functions are represented as trees within a genetic algorithm—a standard genetic programming approach. Trees are a logical choice to repre-

sent functions due to their hierarchical nature. The loss-function search space is defined by the following tree nodes:

Unary Operators: $\log(\circ), \circ^2, \sqrt{\circ}$

Binary Operators: $+, *, -, \div$

Leaf Nodes: $x, y, 1, -1$, where x represents a true label, and y represents a predicted label.

A loss function's fitness within the genetic algorithm is the validation performance of a network trained with that loss function. To expedite the discovery process, and encourage the invention of loss functions that make learning faster, training does not proceed to convergence. First, a fitness of 0 is automatically assigned to trees that do not contain both at least one x and one y. Second, unstable training sessions that result in NaN values are assigned a fitness of 0. Third, fitness values are cached to avoid the need to retrain the same network twice. These cached values are each associated with a canonicalized version of their corresponding tree, resulting in fewer required evaluations.

The initial population is composed of randomly generated trees with a maximum depth of two. Recursively starting from the root, nodes are randomly chosen from the allowable operator and leaf nodes using a weighting (where $\log(\circ), x, y$ are three times as likely and $\sqrt{\circ}$ is two times as likely as $+, *, -, \div, 1, -1$). This weighting can impart a bias and prevent, for example,

the integer 1 from occurring too frequently. The genetic algorithm typically has a population size of 80, incorporates elitism with six elites per generation, and uses roulette sampling.

As is typical in genetic programming, recombination is accomplished by randomly splicing two trees together. For a given pair of parent trees, a random element is chosen in each as a crossover point. The two subtrees, whose roots are the two crossover points, are then swapped with each other. Figure 4.1 presents an example of this method of recombination. Both resultant trees become part of the next generation. Recombination occurs with a probability of 80% for a pair of parent trees.

To introduce variation into the population, the genetic algorithm has the following mutations, applied in a bottom-up fashion (i.e., deeper tree nodes are mutated earlier):

- Integer scalar nodes are incremented or decremented with a 5% probability.
- Nodes are replaced with a weighted-random node with the same number of children with a 5% probability.
- Nodes (and their children) are deleted and replaced with a weighted-random leaf node with a $5\% * 50\% = 2.5\%$ probability.
- Leaf nodes are deleted and replaced with a weighted-random element (and weighted-random leaf children if necessary) with a $5\%*50\% = 2.5\%$

probability.

Mutations, as well as recombination, allow for trees of arbitrary depth to be evolved. Thus, GLO can discover arbitrarily complex functions as needed. GLO populations' individuals do tend to grow in size as evolution progresses, but this is not an issue in practice.

Combined, the iterative sampling, recombination, and mutation of trees within the population leads to the discovery of new loss functions which maximize fitness.

4.1.2 Coefficient Optimization

Loss functions found by the above genetic algorithm can all be thought of as having unit coefficients for each node in the tree. This set of coefficients can be represented as a vector with dimensionality equal to the number of nodes in a loss function's tree. The number of coefficients can be reduced by pruning away coefficients that can be absorbed by others (e.g., $3\ (5x + 2y) = 15x + 6y$). In GLO, the coefficient vector is optimized independently and iteratively using a covariance-matrix adaptation evolutionary strategy (CMA-ES) [65].

CMA-ES is a popular population-based, black-box optimization technique for rugged, continuous spaces. CMA-ES functions by maintaining a covariance matrix around a mean point that represents a distribution of solutions. At each generation, CMA-ES adapts the distribution to better fit

evaluated objective values from sampled individuals. In this manner, the area in the search space which is being sampled at each step dynamically grows, shrinks, and moves as needed to maximize sampled candidates' fitnesses.

The specific variant of CMA-ES that GLO uses is $(\mu/\mu, \lambda)$-CMA-ES [66], which incorporates weighted rank-μ updates [64] to reduce the number of objective function evaluations needed. The implementation of GLO presented in this chapter uses an initial step size $\sigma = 1.5$. As in the discovery phase, the objective function is the network's performance on a validation dataset after a shortened training period.

4.1.3 Experiments

Since Lossferatu (described in Section 3.3) had not yet been developed and candidate evaluation was a relatively fast process, GLO was implemented in a bespoke system whereby training was distributed across the network to a cluster of dedicated machines, using HTCondor [170] for scheduling. Each machine in this cluster had one NVIDIA GeForce GTX Titan Black GPU and two quad-core Intel Xeon E5-2603 CPUs running at a base frequency of 1.80GHz with 8GB of memory. Training itself wass implemented with TensorFlow [7] in Python. The primary components of GLO (i.e., the genetic algorithm and CMA-ES) were implemented in Swift. These components run centrally on one machine and asynchronously dispatch work to the Condor cluster over SSH. The SwiftCMA [49] and SwiftGenetics [50] were developed and open-sourced as part of this implementation.

GLO experiments were run on the basic CNN architecture described in Section 3.2.1, with the MNIST image classification benchmark dataset. Such experiments were run for up to 100 generations, and stopped when they qualitatively appeared to have converged, i.e. subsequent generations' top-candidate performance plateaued. Models in candidate evaluations were trained for 2,000 steps (i.e., 10% of full training duration).

4.2 Discovering Baikal

The best loss function that was discovered by a run of GLO with the MNIST dataset is a novel function, the Baikal loss, named for its similarity to the bathymetry of Lake Baikal when plotted (Section 4.3). Compared to the cross-entropy loss, Baikal trained models more quickly while converging to higher accuracies. Additionally, Baikal performs better than cross-entropy when using few training samples. Baikal's advantages also transfer to more complex datasets, as will be shown below.

Baikal is defined as

$$\mathcal{L}_{\text{Baikal}} = -\frac{1}{n}\sum_{i=0}^{n}\log(y_i) - \frac{x_i}{y_i}, \tag{4.1}$$

where $\mathbf{x}$ is a sample from the true distribution, $\mathbf{y}$ is a sample from the predicted distribution, and n is the number of classes. Baikal was discovered from a single run of GLO. Additionally, after coefficient optimization, GLO arrived at the following version of the Baikal loss:.

$$\mathcal{L}_{\text{BaikalCMA}} = -\frac{1}{n}\sum_{i=0}^{n} c_0\left(c_1 * \log(c_2 * y_i) - c_3\frac{c_4 * x_i}{c_5 * y_i}\right), \tag{4.2}$$

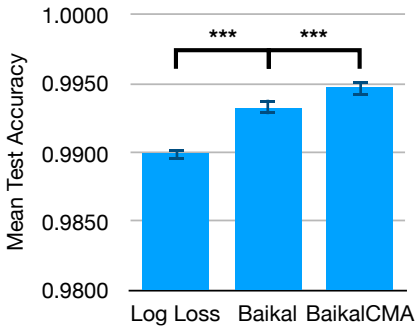


Figure 4.2: **Mean testing accuracy on the MNIST image classification benchmark, from ten independent runs each for cross-entropy, Baikal, and BaikalCMA loss functions.** Both Baikal and BaikalCMA loss functions provide statistically significant improvements to testing accuracy over the cross-entropy loss.

where $c_0 = 2.7279, c_1 = 0.9863, c_2 = 1.5352, c_3 = -1.1135, c_4 = 1.3716, c_5 = -0.8411$.

This loss function, BaikalCMA, was selected from the ninth generation of a 35 generation run of CMA-ES for having the highest validation accuracy out of the population. The Baikal and BaikalCMA loss functions had validation accuracies at $2,000$ steps equal to 0.9838 and 0.9902, respectively. For comparison, at $2,000$ steps, the cross-entropy loss had a validation accuracy of 0.9700. Models trained with the Baikal loss on MNIST and CIFAR-10 (to test transfer) are the primary vehicle for validating GLO's efficacy, as detailed in subsequent sections.

Figure 4.2 shows the increase in testing accuracy that Baikal and BaikalCMA provide on MNIST over models trained with the cross-entropy loss. Over 10 trained models each, the mean testing accuracies for cross-entropy loss, Baikal,

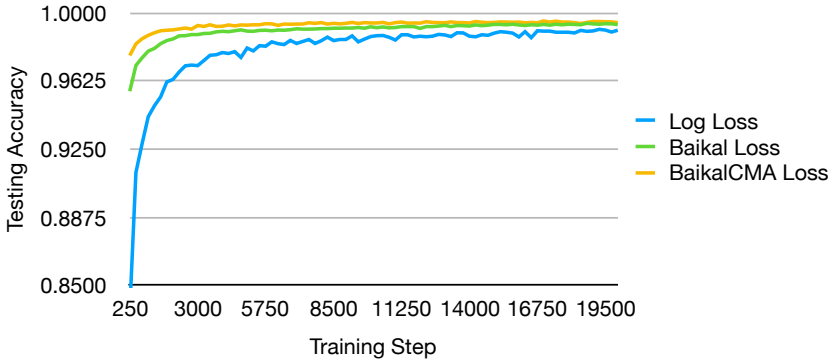


Figure 4.3: **Training curves for different loss functions on MNIST.** Baikal and BaikalCMA result in faster and smoother training compared to the cross-entropy loss.

and BaikalCMA were 0.9899, 0.9933, and **0.9947**, respectively. This increase is statistically significant, with a p-value of 2.4×10^{-11}, in a Welch's t-Test as described in Section 3.4, with 10 samples from each distribution. With the same significance test, the increase in accuracy from BaikalCMA over Baikal is statistically significant as well, with a p-value of 8.5045×10^{-6}. Therefore, loss-function metalearning with GLO is shown to be an effective way to increase model performance, with both evolution phases contributing.

Training curves for networks trained with the cross-entropy loss, Baikal, and BaikalCMA are shown in Figure 4.3. Each curve represents 80 testing dataset evaluations spread evenly (i.e., every 250 steps) throughout 20,000 steps of training on MNIST. Networks trained with Baikal and BaikalCMA both learn significantly faster than the cross-entropy loss. These phenomena make Baikal a compelling loss function for fixed time-budget training, where the improvement in resultant accuracy over the cross-entropy loss becomes

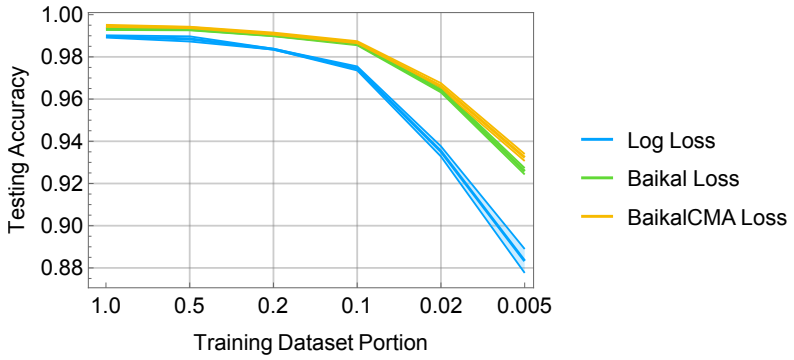


Figure 4.4: **Sensitivity to different dataset sizes for different loss functions on MNIST.** For each size, five experiments were run. Standard deviations are presented as error bands Baikal and BaikalCMA increasingly outperform the cross-entropy loss on small datasets, providing evidence of reduced overfitting.

most evident.

4.2.1 Training Data Requirements

Figure 4.4 provides an overview of the effects of dataset size on networks trained with cross-entropy loss, Baikal, and BaikalCMA. For each training dataset portion size, five individual networks were trained for each loss function. On the 0.05 dataset portion (i.e., the smallest, with only $2,750$ training samples), the cross-entropy loss frequently exhibited numerical instability. Thus, these specific experiments had to be run many times to yield five fully-trained networks. Notably, Baiikal and BaikalCMA did not yield unstable training runs and no other dataset portions exhibited instability with the cross-entropy loss. As in previous experiments, MNIST networks were trained for 20,000 steps.

The degree by which Baikal and BaikalCMA outperform cross-entropy loss increases as the training dataset becomes smaller. This observation provides evidence that networks overfit less when they are trained with Baikal or BaikalCMA. As expected, BaikalCMA outperforms Baikal at all tested dataset sizes. The size of this improvement in accuracy does not grow as significantly as the improvement over cross-entropy loss, leading to the belief that the data fitting characteristics of Baikal and BaikalCMA are similar.

Overall, these reduced data requirements allow small datasets to be used more effectively. This finding has practical ramifications, as not many datasets found in the world are large. Going forward, custom loss functions could be evolved to target these small datasets specifically.

4.2.2 Loss Function Transfer to CIFAR-10

Figure 4.5 presents a collection of 18 separate tests of the cross-entropy loss and Baikal applied to CIFAR-10. Baikal is found to outperform cross-entropy across all training durations, with the difference being more prominent in early training. These results present an interesting use case for GLO, where a loss function that is found on a simpler dataset can be transferred to a more complex dataset while still maintaining performance improvements. In effect, Baikal allows for faster training, which supports that GLO loss functions could be particularly useful in fixed time-budget scenarios.

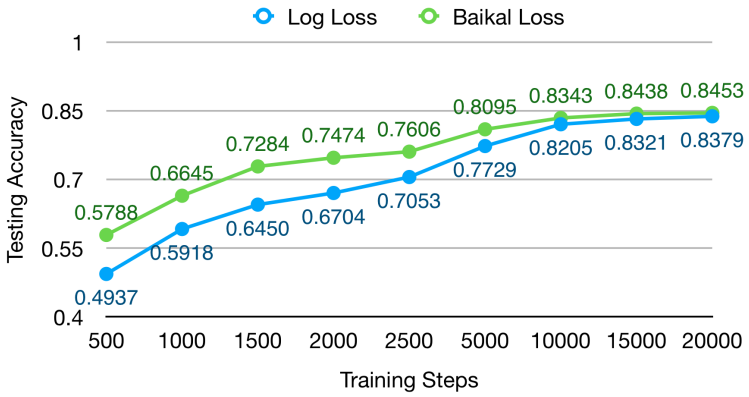


Figure 4.5: **Testing accuracy across varying training steps on the CIFAR-10 image classification benchmark.** The Baikal loss, which has been transferred from MNIST, outperforms the cross-entropy loss on all training durations. This effect is more pronounced early in training, suggesting that Baikal results in faster training.

4.3 What makes Baikal work?

This section presents a conceptual analysis of the Baikal loss function, followed by experiments to elucidate why Baikal works better than the cross-entropy loss. The conclusion is that Baikal results in implicit regularization, reducing overfitting.

Loss functions used on the MNIST dataset, a 10-dimensional classification problem, are difficult to plot and visualize graphically. To simplify, they are analyzed in this section in the context of binary classification, with $n = 2$, the Baikal loss expands to

$$\mathcal{L}_{\text{Baikal2D}} = -\frac{1}{2}\left(\log(y_0) - \frac{x_0}{y_0} + \log(y_1) - \frac{x_1}{y_1}\right). \tag{4.3}$$

Since vectors $\mathbf{x}$ and $\mathbf{y}$ sum to 1, by consequence of being passed through a softmax function, for binary classification $\mathbf{x} = \langle x_0, 1 - x_0 \rangle$ and $\mathbf{y} = \langle y_0, 1 - y_0 \rangle$.

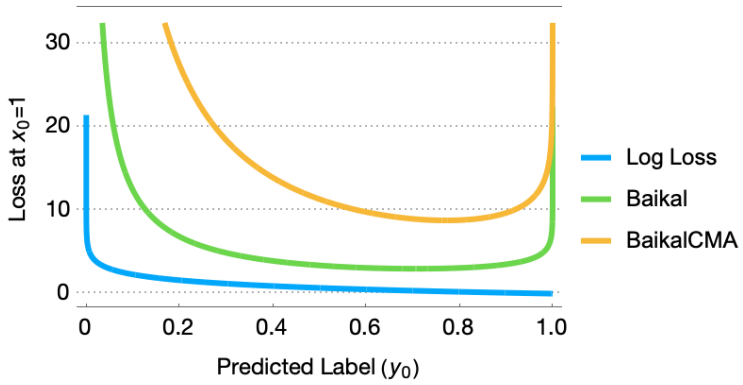


Figure 4.6: **Binary classification loss functions plotted at $x_0 = 1$.** Correct predictions lie on the right side of the graph, and incorrect ones on the left. The log loss decreases monotonically, while Baikal and BaikalCMA present counterintuitive, sharp increases in loss as predictions approach the true label. This phenomenon provides regularization by preventing the model from being too confident in its predictions.

This constraint simplifies the binary Baikal loss to a function of two variables (x_0 and y_0),

$$\mathcal{L}_{\text{Baikal2D}} \propto -\log(y_0) + \frac{x_0}{y_0} - \log(1 - y_0) + \frac{1 - x_0}{1 - y_0}. \tag{4.4}$$

This same methodology can be applied to the cross-entropy loss and BaikalCMA, and plotted in Figure 4.6.

In practice, true labels are assumed to be correct with certainty, thus, x_0 is equal to either 0 or 1. Figure 4.6 shows the specific case where $x_0 = 1$. The cross-entropy loss is monotonically decreasing, while the Baikal and BaikalCMA loss functions counterintuitively show an increase in loss as the predicted label, y_0, approaches the true label x_0. This unexpected increase allows the loss functions to prevent the model from becoming too confident in

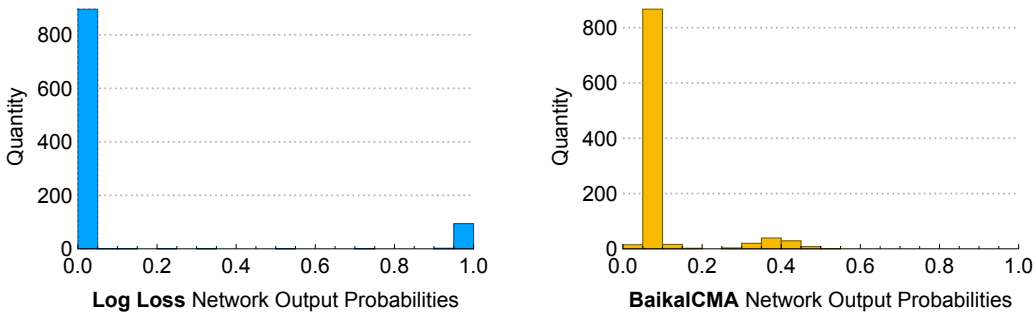


Figure 4.7: **Outputs of networks trained with cross-entropy loss and BaikalCMA.** With BaikalCMA, the peaks are shifted away from extreme values and are more spread out, indicating implicit regularization. The BaikalCMA histogram matches that from a network trained with a confidence regularizer [133], supporting the hypothesis that GLO can discover loss functions that bias training in such a way that results in regularization.

its output predictions, thus providing a form of regularization.

The Baikal and BaikalCMA loss functions are surprising in that they incur a high loss when the output is very close to the correct value (as illustrated in Figure 4.6). Although at first glance this behavior is counterintuitive, it may provide an important advantage. The outputs of a trained network will not be exactly correct, although they are close, and therefore the network is less likely to overfit. Thus, these loss functions provide an implicit form of regularization, enabling better generalization.

This effect is similar to that of the confidence regularizer [133], which penalizes low-entropy prediction distributions. The bimodal distribution of outputs that results from confidence regularization is nearly identical to that of a network trained with BaikalCMA. Note that while these outputs are

typically referred to as probabilities in the literature, this is often an innacurate interpretation [45], i.e. a model can produce highly-uncertain predictions while having scaled target logits with values close to 1.

Histograms of these distributions on the test dataset for cross-entropy and BaikalCMA networks, after 15,000 steps of training on MNIST, are shown in Figure 4.7. The abscissae in Figures 4.6 and 4.7 match, making it clear how the distribution for BaikalCMA has shifted away from extreme values. The improved behavior under small-dataset conditions described in Section 4.2.1 provides further evidence for implicit regularization; less overfitting was observed when using Baikal and BaikalCMA compared to the cross-entropy loss.

As also seen in Figure 4.6, the minimum for the Baikal loss lies near $y_0 = 0.71$, while the minimum for the BaikalCMA loss lies near $y_0 = 0.77$. This minimum, along with the more pronounced slope around $x_0 = 0.5$, is likely a reason why BaikalCMA performs better than Baikal. The greater slope over a wider domain of the function implies that there are larger gradients over a wider domain as well, a property which can lead to faster training. The different minima can be thought of as a tuning of the point at which prediction confidence is penalized.

As was detailed in Section 3.2.1, MNIST networks were trained with dropout [75], and CIFAR-10 networks with L_2 weight decay and local response normalization [95]. Yet Baikal was able to improve performance further. Thus, the implicit regularization provided by Baikal and BaikalCMA complements the different types of regularization already present in the trained networks

and they can be combined to achieve an increased effect.

4.4 Expanded Search Space

The search space for GLO can be extended to include a network's unscaled logits (i.e., the output of a classification neural network before the softmax layer) as a potential leaf node. The addition of unscaled logits extends the base implementation of GLO to support loss functions that take three variables, rather than two (i.e., ground-truth labels and scaled logits). Conceptually, the availability of more information should allow training to proceed in a more intelligent manner. Unscaled logits in particular can provide information on the network's raw, unnormalized outputs.

When running GLO with this expanded search space on MNIST, a new loss function, referred to as the FastLogit loss was discovered:

$$\mathcal{L}_{\text{FastLogit}} = -\frac{1}{n}\sum_{i=0}^{n}(\tilde{y}_i - x_i) * \left(\frac{x_i}{y_i} - (\tilde{y}_i - (-2))\right) , \tag{4.5}$$

where $\tilde{\mathbf{y}}$ are the network's unscaled logits. Notably, this loss function is more complex than Baikal or the log loss, showing how evolution can take advantage of expanded search spaces.

True to its name, the FastLogit loss is able to learn more quickly than Baikal, while converging to a comparable accuracy (shown in Figure 4.8). This improvement over Baikal provides support for having search spaces that include more types of data.

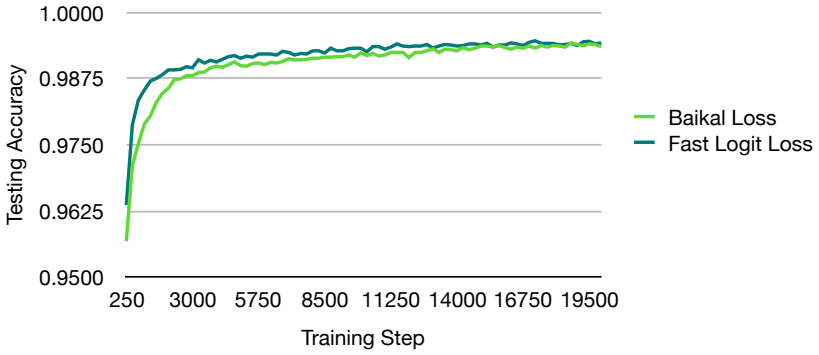


Figure 4.8: **Training curves for the FastLogit and Baikal loss functions on MNIST.** FastLogit results in faster training compared to Baikal, with a comparable final accuracy.

4.5 Discussion

This chapter developed a first approach to loss function metalearning by introducing an evolutionary computation approach to it. GLO was evaluated experimentally on image classification domains, and discovered a surprising new loss function, Baikal. Experiments showed substantial improvements in accuracy, convergence speed, and data requirements when using Baikal. Further analysis suggested that these improvements result from implicit regularization that reduces overfitting to the data. This regularization complements the existing regularization in trained networks. The reasons for this regularization effect's existence, and its specific behavior, are covered in more detail in Chapter 6.

GLOs tree-based representation provides a unique amount of flexibility that can represent any function given a set of operators and tokens. This was leveraged by incorporating a new input into the loss-function search space:

unscaled logits. With this expanded search space, GLO discovered the Fast-Logit loss function. This loss function was able to learn more quickly than the Baikal loss function, while converging to a comparable level of performance.

Expanding this search space further, in the future, it may be advantageous to include preexisting regularization terms, such as an L_2 weight decay term, as optional leaf nodes. An alternative embodiment to this idea may be to forgo including regularization terms in GLO trees, and instead include them by default, with a coefficient of 0, in all loss functions, letting their importance be determined during the coefficient optimization step. Forgoing more freeform optimization of regularization terms as tree nodes would reduce the difficulties associated with larger search spaces in evolution.

While the flexibility that tree-based representations provide makes it possible to discover of unique loss functions, unbridled flexibility may be too much in practice, for the majority of use cases. GLO operates on a space where exploration is very favored, and exploitation is difficult; there is a lot of serendipity involved in the search process. The majority of loss-function candidates, particularly in early generations, have degenerate characteristics and are not even able to train neural networks. Furthermore, individual mutations on the tree tend to have a large impact on the function (e.g., swapping a multiplication node with a division node), making the fitness landscape rugged. To accommodate these deleterious properties, large populations and many generations are required for evolution to function. In a world where there are tangible costs associated with evaluating several thousands of neural networks,

applying GLO to more sophisticated models may be out of reach.

This excess flexibility is further complicated by GLO's two-phase approach to discovering and optimizing loss functions. An optimal loss function with coefficients may not necessarily be a good loss function if those coefficients are set to unit values. More concisely: the structural evolution and coefficient evolution steps may clash, and thus certain loss functions will remain out-of-reach to GLO.

A thoughtfully designed representation, with a smoother fitness landscape, and where efficacious loss functions are less sparsely scattered in the search space, can more easily and expeditiously guide evolution towards good candidates. This is the topic of the next chapter.

4.6 Conclusion

This chapter proposed Genetic Loss-function Optimization (GLO) as a general framework for discovering and optimizing loss functions for a given task. A surprising new loss function, Baikal, was discovered in the experiments, and shown to outperform the cross-entropy loss on MNIST and CIFAR-10 in terms of accuracy, training speed, and data requirements. Further analysis suggested that Baikal's improvements result from implicit regularization that reduces overfitting to the data. While GLO's search space provides immense flexibility, perhaps a more refined search space and a single-phase search can result in a more computationally efficient, practical technique. Such a technique, TaylorGLO, is presented in the next chapter.

Chapter 5

TaylorGLO

In Chapter 4, loss-function discovery and optimization were tackled as a new type of metalearning with the development of Genetic Loss-function Optimization (GLO). Focusing on a neural network's root training goal, it aims to discover better ways to define what is being optimized. However, these loss functions can be challenging to optimize because they have a discrete nested structure as well as continuous coefficients. In an ideal case, loss functions would be mapped into arbitrarily long, fixed-length vectors in a Hilbert space. This mapping should be smooth, well-behaved, well-defined, incorporate both a function's structure and coefficients, and should by its very nature exclude large classes of infeasible loss functions.

This chapter introduces such an approach: *Multivariate Taylor expansion-based genetic loss-function optimization* (TaylorGLO). Loss functions discovered by TaylorGLO outperform the standard cross-entropy loss (or log loss), as well as the Baikal loss discovered by the original GLO technique, on a variety of datasets with several different network architectures. TaylorGLO thus further establishes loss-function optimization as a promising new direction for metalearning.

5.1 Multivariate Taylor Expansions

Taylor expansions [169] are a well-known function approximator that can represent differentiable functions within the neighborhood of a point using a polynomial series. Below, the common univariate Taylor expansion formulation is presented, followed by a natural extension to the multivariate case, i.e. to arbitrarily-multivariate functions. Multivariate Taylor expansions provide the basis for TaylorGLO's loss function parameterization.

Consider a given real-valued function $f(x) : \mathbb{R} \to \mathbb{R}$ that is $C^{k_{\max}}$ smooth (i.e., first through $k_{\max}$ derivatives are continuous). A kth-order Taylor approximation for this function, $f(x)$, at point $a \in \mathbb{R}$, $\hat{f}_k(x, a)$, where $0 \leq k \leq k_{\max}$, can be constructed as

$$\hat{f}_k(x, a) = \sum_{n=0}^{k} \frac{1}{n!} f^{(n)}(a)(x - a)^n. \tag{5.1}$$

Conventional, univariate Taylor expansions have a natural extension to arbitrarily high-dimensional inputs of f. Given a $C^{k_{\max+1}}$ smooth, real-valued function $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$, a kth-order Taylor approximation at point $\mathbf{a} \in \mathbb{R}^n$, $\hat{f}_k(\mathbf{x}, \mathbf{a})$, where $0 \leq k \leq k_{\max}$, can be constructed analogous to Equation 5.1. The stricter smoothness constraint compared to the univariate case allows for the application of Schwarz's theorem on equality of mixed partials, obviating the need to take the order of partial differentiation into account.

More specirically, let an nth-degree multi-index, $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ be defined, where $\alpha_i \in \mathbb{N}_0$, $|\alpha| = \sum_{i=1}^{n} \alpha_i$, $\alpha! = \prod_{i=1}^{n} \alpha_i!$. $\mathbf{x}^\alpha = \prod_{i=1}^{n} x_i^{\alpha_i}$, and $\mathbf{x} \in \mathbb{R}^n$. Multivariate partial derivatives can be concisely written using the

multi-index as

$$\partial^\alpha f = \partial_1^{\alpha_1} \partial_2^{\alpha_2} \cdots \partial_n^{\alpha_n} f = \frac{\partial^{|\alpha|}}{\partial x_1^{\alpha_1} \partial x_2^{\alpha_2} \cdots \partial x_n^{\alpha_n}}. \tag{5.2}$$

Thus, discounting the remainder term, the multivariate Taylor expansion for $f(\mathbf{x})$ at $\mathbf{a}$ is

$$\hat{f}_k(\mathbf{x}, \mathbf{a}) = \sum_{\forall \alpha, |\alpha| \leq k} \frac{1}{\alpha!} \partial^\alpha f(\mathbf{a})(\mathbf{x} - \mathbf{a})^\alpha. \tag{5.3}$$

The unique partial derivatives in $\hat{f}_k$ and $\mathbf{a}$ are parameters for a kth order Taylor expansion. Thus, a kth order Taylor expansion of a function in n variables requires n parameters to define the center, $\mathbf{a}$, and one parameter for each unique multi-index α, where $|\alpha| \leq k$. That is,

$$\#_{\text{parameters}}(n, k) = n + \binom{n+k}{k} = n + \frac{(n+k)!}{n!\,k!}\ . \tag{5.4}$$

The multivariate Taylor expansion can be leveraged for a novel loss-function parameterization. It enables TaylorGLO, a way to efficiently optimize loss functions, as will be described in subsequent sections.

5.2 Loss Functions as Multivariate Taylor Expansions

Let an n-class classification loss function be defined as:

$$\mathcal{L}_f(\mathbf{x}, \mathbf{y}) = -\frac{1}{n} \sum_{i=1}^{n} f(x_i, y_i). \tag{5.5}$$

The function $f(x_i, y_i)$ can be replaced by its kth-order, bivariate Taylor expansion, $\hat{f}_k(x, y, a_x, a_y)$. More sophisticated loss functions can be supported

by having more input variables, beyond x_i and y_i, such as a time variable or unscaled logits. This approach can be useful, for example, to evolve loss functions that change as training progresses.

For example, a loss function in $\mathbf{x}$ and $\mathbf{y}$ has the following 3rd-order parameterization with parameters θ (where $\mathbf{a} = \langle \theta_0, \theta_1 \rangle$):

$$\mathcal{L}(\mathbf{x}, \mathbf{y}) = -\frac{1}{n} \sum_{i=1}^{n} \Big[\theta_2 + \theta_3(y_i - \theta_1) + \tfrac{1}{2}\theta_4(y_i - \theta_1)^2 + \tfrac{1}{6}\theta_5(y_i - \theta_1)^3 + \theta_6(x_i - \theta_0) + \theta_7(x_i - \theta_0)(y_i - \theta_1) + \tfrac{1}{2}\theta_8(x_i - \theta_0)(y_i - \theta_1)^2 + \tfrac{1}{2}\theta_9(x_i - \theta_0)^2 + \tfrac{1}{2}\theta_{10}(x_i - \theta_0)^2(y_i - \theta_1) + \tfrac{1}{6}\theta_{11}(x_i - \theta_0)^3 \Big] \tag{5.6}$$

Notably, the reciprocal-factorial coefficients can be integrated to be a part of the parameter set by direct multiplication if desired.

As will be shown in this chapter, the representation technique makes it possible to train neural networks that are more accurate and learn faster, than those with tree-based loss function representations. This result is due to several useful properties of this Taylor expansion approach:

- It guarantees smooth functions;
- Functions do not have poles (i.e., discontinuities going to infinity or negative infinity) within their relevant domain;
- It can be implemented purely as compositions of addition and multiplication operations;
- It can be trivially differentiated;

- Nearby points in the search space yield similar results (i.e., the search space is locally smooth), making the fitness landscape easier to search;
- Valid loss functions can be found in fewer generations and with higher frequency, allowing for smaller population sizes to be used;
- Loss function discovery is consistent and not dependent on a specific initial population; and
- The search space has a tunable complexity parameter, i.e. the order of the expansion.

These properties are not necessarily held by alternative function approximators. For instance:

Fourier series are well suited for approximating periodic functions [41]. Therefore, they are not as well suited for loss functions, whose local behavior within a narrow domain is important. Being a composition of waves, Fourier series tend to have many critical points within the domain of interest. Gradients fluctuate around such points, making gradient descent infeasible. Additionally, close approximations require a large number of terms, which in itself can be injurious, causing large, high-frequency fluctuations known as "ringing", due to Gibb's phenomenon [190].

Padé approximants can be more accurate approximations than Taylor expansions; indeed, Taylor expansions are thus a special case with $M = 0$

[58]. However, unfortunately Padé approximants can model functions with one or more poles, which valid loss functions typically should not have. These problems still exist, and are exacerbated, for Chisholm approximants [28] (a bivariate extension of Padé approximants) and Canterbury approximants [59] (a multivariate generalization of Padé approximants).

Laurent polynomials can represent functions with discontinuities, the simplest being x^{-1}. While Laurent polynomials provide a generalization of Taylor expansions into negative exponents, the extension is not useful because it leads to the same issues as Padé approximants.

Polyharmonic splines seem like an excellent fit since they can represent continuous functions within a finite domain. However, the number of parameters is prohibitive in the multivariate case.

Lagrange polynomials interpolate between a set of control points using a weighted sum of basis polynomials [188]. While Runge's phenomenon [144] is minimized and a number of parameters can be elimited by restricting control points to Chebyshev nodes (i.e., the roots of Chebyshev polynomials [25]), the number of parameters may still be prohibitive in the multivariate case, as with polyharmonic splines, and additionally requires a discrete selection of Chebyshev nodes.

The multivariate Taylor expansion is therefore a better choice than

the alternatives. It makes it possible to optimize loss functions efficiently in TaylorGLO, as will be described next.

5.3 The TaylorGLO Approach

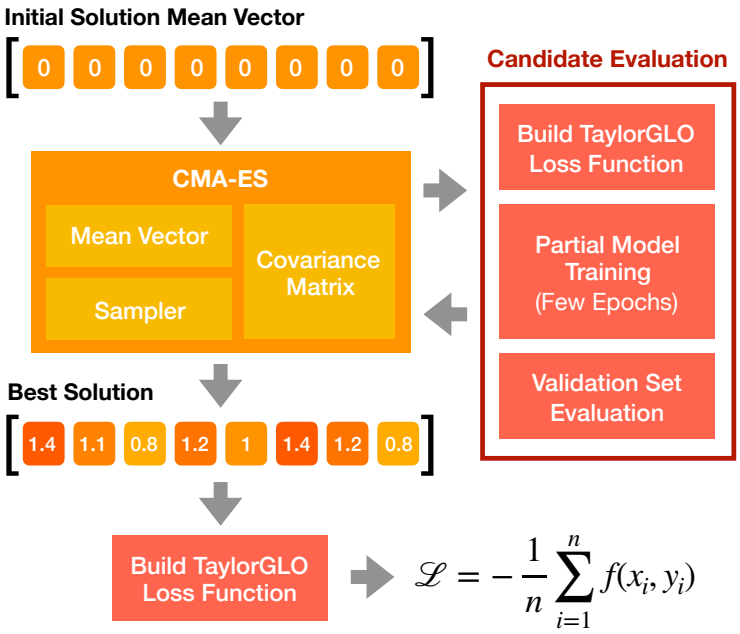


Figure 5.1: **The TaylorGLO approach.** Loss functions are represented as vectors that encode parameters to the TaylorGLO loss function parameterization. Starting with an initial unbiased mean solution, CMA-ES iteratively maximizes the validation accuracy—or any other non-differentiable fitness metric—that results from training with TaylorGLO loss function candidates. CMA-ES maintains a mean vector and corresponding covariance matrix which are used to sample candidates at each generation. The candidate with the highest fitness is chosen as the final, best solution from TaylorGLO.

TaylorGLO (Figure 5.1) aims to find the optimal parameters for a loss function parameterized as a multivariate Taylor expansion, as described in

Section 5.2. The parameters for a Taylor approximation (i.e., the center point and partial derivatives) are referred to as $\theta_{\hat{f}}$, where $\theta_{\hat{f}} \in \Theta$, $\Theta = \mathbb{R}^{\#\text{parameters}}$. TaylorGLO strives to find the vector $\theta^*_{\hat{f}} \in \Theta$ that parameterizes the optimal loss function for a task. Because the values are continuous, as opposed to discrete graphs of the original GLO, it is possible to use continuous optimization methods, such as CMA-ES.

Like in GLO, the specific variant of CMA-ES that TaylorGLO uses is $(\mu/\mu, \lambda)$-CMA-ES [66], which incorporates weighted rank-μ updates [64] to reduce the number of objective function evaluations that are needed.

In contrast with GLO, in TaylorGLO, CMA-ES is used to find $\theta^*_{\hat{f}}$. At each generation, CMA-ES samples points in Θ whose fitness is determined; this fitness evaluation is accomplished by training a model with the corresponding loss function and evaluating the model on a validation dataset. Fitness evaluations may be distributed across multiple machines in parallel and retried a limited number of times upon failure (e.g., exploding gradients). An initial vector of $\theta_{\hat{f}} = \mathbf{0}$ is chosen as a starting point in the search space to avoid bias.

Note that fully training a model can prove to be prohibitively expensive in many problems. Fundamentally, there is a positive correlation between performance near the beginning of training and at the end of training. In order to identify the most promising candidates, it is enough to train the models only partially. This type of approximate evaluation is widely done in the field of evolutionary computation [60, 87]. An additional positive effect is that evaluation then favors loss functions that learn more quickly.

For a loss function to be useful, it must have a derivative that depends on the prediction. Therefore, internal terms that do not contribute to $\frac{\partial}{\partial \mathbf{y}} \mathcal{L}_f(\mathbf{x}, \mathbf{y})$ can be trimmed away. This property implies that any term t within $f(x_i, y_i)$, where $\frac{\partial}{\partial y_i} t = 0$, can be replaced with 0.

For example, this refinement simplifies Equation 5.6, providing a reduction in the number of parameters from twelve to eight:

$$\mathcal{L}(\mathbf{x}, \mathbf{y}) = -\frac{1}{n} \sum_{i=1}^{n} \Big[\theta_2(y_i - \theta_1) + \tfrac{1}{2}\theta_3(y_i - \theta_1)^2 + \tfrac{1}{6}\theta_4(y_i - \theta_1)^3 + \theta_5(x_i - \theta_0)(y_i - \theta_1) + \tfrac{1}{2}\theta_6(x_i - \theta_0)(y_i - \theta_1)^2 + \tfrac{1}{2}\theta_7(x_i - \theta_0)^2(y_i - \theta_1) \Big] . \tag{5.7}$$

Loss functions of this form are evolved with TaylorGLO and evaluated in the following section.

5.4 Performance Experiments

This section illustrates the TaylorGLO process and demonstrates how the evolved loss functions can improve performance over the standard cross-entropy loss function, especially on reduced datasets. A summary of results on four datasets across a variety of models is shown in Table 5.1.

5.4.1 The TaylorGLO Discovery Process

Figure 5.2 illustrates the evolution process over 60 generations, which is sufficient to reach convergence on the MNIST dataset. TaylorGLO is able to discover high-performing loss functions quickly, i.e. within 20 generations.

Table 5.1: **Test-set accuracy of loss functions discovered by TaylorGLO compared with that of the cross-entropy loss.** The TaylorGLO results are based on the loss function with the highest validation accuracy during evolution. All averages are from ten separately trained models and p-values are from one-tailed Welch's t-Tests. Standard deviations are shown in parentheses. TaylorGLO discovers loss functions that perform significantly better than the cross-entropy loss in almost all cases, including those that use Cutout. This suggests that it provides a different form of regularization. Accuracies are indicated in bold if statistically significantly higher.

Task and Model	TaylorGLO Acc.	Baseline Acc.	p-value
MNIST			
Basic CNN	**0.9951 (0.0005)**	0.9899 (0.0003)	2.95×10^{-15}
CIFAR-10			
AlexNet	**0.7901 (0.0026)**	0.7638 (0.0046)	1.76×10^{-10}
AlexNet + Cutout	**0.7786 (0.0022)**	0.7741 (0.0040)	0.0049
AlexNet + CutMix	**0.7928 (0.0027)**	0.7856 (0.0026)	8.13×10^{-6}
ResNet-20	0.9136 (0.0029)	0.9146 (0.0019)	0.2021
Pre ResNet-20	**0.9169 (0.0014)**	0.9153 (0.0021)	0.0400
AllCNN-C	**0.9271 (0.0013)**	0.8965 (0.0021)	0.42×10^{-17}
AllCNN-C + Cutout	**0.9329 (0.0022)**	0.8911 (0.0037)	1.60×10^{-14}
AllCNN-C + CutMix	**0.9327 (0.0014)**	0.8749 (0.0042)	1.89×10^{-13}
Wide ResNet 16-8	**0.9558 (0.0011)**	0.9528 (0.0012)	1.77×10^{-5}
Wide ResNet 16-8 + Cutout	**0.9618 (0.0010)**	0.9582 (0.0011)	2.55×10^{-7}
Wide ResNet 28-5	0.9548 (0.0015)	0.9556 (0.0011)	0.0984
Wide ResNet 28-5 + Cutout	0.9621 (0.0013)	0.9616 (0.0011)	0.1882
CIFAR-100			
PyramidNet 110a48	0.7409 (0.0040)	**0.7523 (0.0037)**	3.87×10^{-6}
PyramidNet 110a48 + Cutout	**0.7708 (0.0029)**	0.7674 (0.0036)	0.0189
SVHN			
Wide ResNet 16-8	**0.9658 (0.0007)**	0.9597 (0.0006)	1.94×10^{-13}
Wide ResNet 16-8 + Cutout	**0.9714 (0.0010)**	0.9673 (0.0008)	9.10×10^{-9}
Wide ResNet 28-5	**0.9657 (0.0009)**	0.9634 (0.0006)	6.62×10^{-6}
Wide ResNet 28-5 + Cutout	**0.9727 (0.0006)**	0.9709 (0.0006)	2.96×10^{-6}

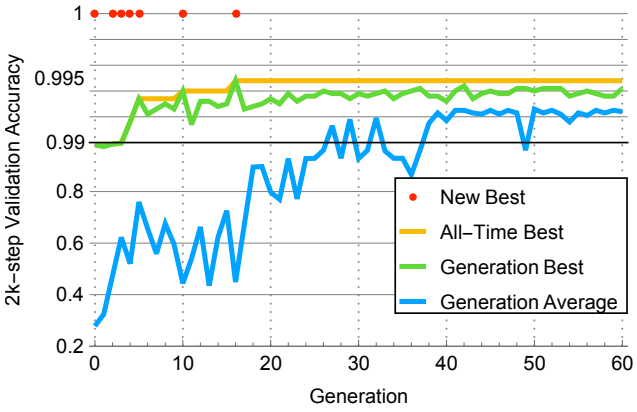


Figure 5.2: **The process of TaylorGLO discovering loss functions in MNIST.** Red dots mark generations where new improved loss functions were found. TaylorGLO discovers good functions in very few generations. The best function had a 2000-step validation accuracy of 0.9948, compared to 0.9903 with the cross-entropy loss, averaged over ten runs. This difference translates to a similar improvement on the test set, as shown in Table 5.1.

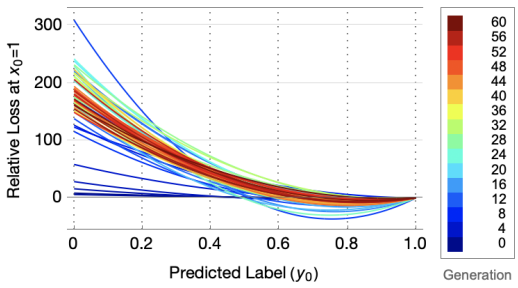
Generations' average validation accuracy approaches generations' best accuracy as evolution progresses, indicating that the population as a whole is improving. Whereas GLO's unbounded search space often results in pathological functions, every TaylorGLO training session completed successfully without any instabilities.

Figure 5.3 shows the shapes and parameters of each generation's highest-scoring loss function. In Figure 5.3*a* the functions are plotted as if they were being used for binary classification, i.e. the loss for an incorrect label on the left and for a correct one on the right (as detailed in Section 4.3). The functions have a distinct pattern through the evolution process. Early generations include a wide variety of shapes, which later converge towards curves with a

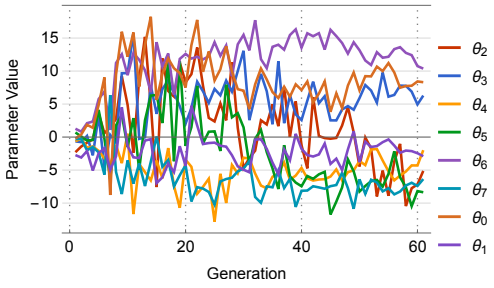
shallow minimum around $y_0 = 0.8$. In other words, the loss increases near the correct output—as it was with the Baikal loss function. Like Baikal, this shape is strikingly different from the cross-entropy loss, which decreases monotonically from left to right, as one might expect all loss functions to do. The evolved shape is effective most likely for the same reason as Baikal. It provides an implicit regularization effect: It discourages the model from outputting unnecessarily extreme values for the correct class, and therefore makes overfitting less likely. This finding again demonstrates the power of machine learning to create innovations beyond human design.

5.4.2 Comparison to GLO

Over 10 fully-trained models, the best TaylorGLO loss function achieved a mean testing accuracy of **0.9951** (stddev 0.0005) in MNIST. In comparison, the cross-entropy loss only reached 0.9899 (stddev 0.0003), and the BaikalCMA loss function discovered by GLO, 0.9947 (stddev 0.0003); both differences are statistically significant (Figure 5.4). Notably, TaylorGLO achieved this result with significantly fewer generations and smaller populations. GLO required 11,120 partial evaluations (i.e., 100 individuals over 100 GP generations plus 32 individuals over 35 CMA-ES generations), while the top TaylorGLO loss function only required **448** partial evaluations, i.e. 4.03% as many. Thus, TaylorGLO achieves improved results with significantly fewer evaluations than GLO.

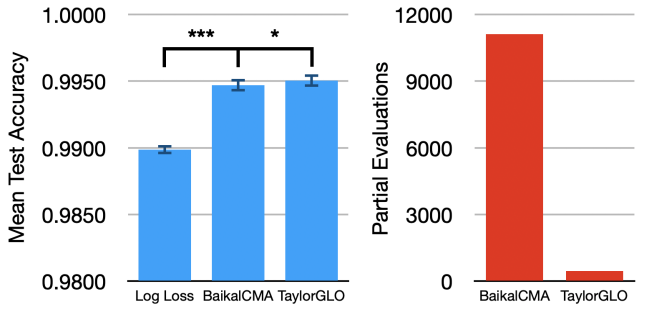


(a) Best discovered functions over time



(b) Best function parameters over time

Figure 5.3: **The best loss functions** (a) **and their respective parameters** (b) **from each generation of TaylorGLO on MNIST.** The functions are plotted in a binary classification modality, showing loss for different values of the network output (y_0 in the horizontal axis) when the correct label is 1.0. The functions are colored according to their generation from blue to red, and vertically shifted such that their loss at $y_0 = 1$ is zero (the raw value of a loss function is not relevant; the derivative, however, is). TaylorGLO explores varying shapes of solutions before narrowing down on functions in the red band. This process can also be seen in (b), where parameters become more consistent over time, and in the population plot shown in Figure 5.2, where fitness plateaus. The final functions decrease from left to right, but have a significant increase in the end. This shape is likely to prevent overfitting during learning, which leads to the observed improved accuracy.



(*a*) Accuracy (*b*) Evaluations

Figure 5.4: (*a*) Mean test accuracy across ten runs on MNIST. The TaylorGLO loss function with the highest validation score significantly outperforms the cross-entropy loss ($p = 2.95 \times 10^{-15}$ in a one-tailed Welch's t-test) and the BaikalCMA loss [53] ($p = 0.0313$). (*b*) Required partial training evaluations for GLO and TaylorGLO on MNIST. The TaylorGLO loss function was discovered with 4% of the evaluations that GLO required to discover BaikalCMA.

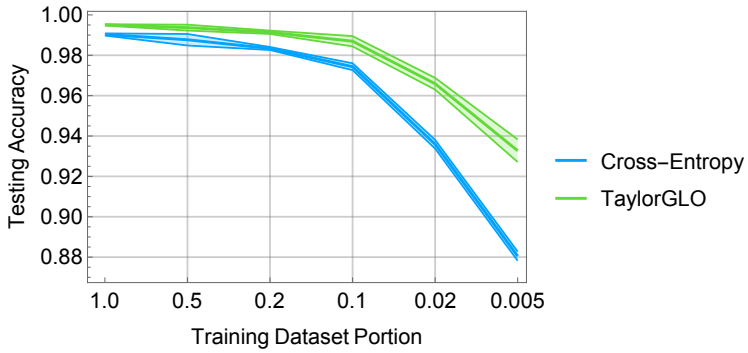


Figure 5.5: Accuracy with reduced portions of the MNIST dataset. Progressively smaller portions of the dataset were used to train the models (averaging over ten runs). The TaylorGLO loss function provides significantly better performance than the cross-entropy loss on all training dataset sizes, and particularly on the smaller datasets. Thus, its ability to discourage overfitting is particularly useful in applications where only limited data is available.

5.4.3 Performance on Reduced Datasets

The performance improvements that TaylorGLO provides are especially pronounced with reduced datasets. For example, Figure 5.5 compares accuracies of models trained for 20,000 steps on different portions of the MNIST dataset (similar results were obtained with other datasets and architectures). Overall, TaylorGLO significantly outperforms the cross-entropy loss. When evolving a TaylorGLO loss function and training against 10% of the training dataset, with 225 epoch evaluations, TaylorGLO reached an average accuracy across ten models of **0.7595** (stddev 0.0062). In contrast, only four out of ten cross-entropy loss models trained successfully, with those reaching a lower average accuracy of 0.6521. Thus, customized loss functions can be especially useful in applications where only limited data is available to train the models, presumably because they are less likely to overfit to the small number of examples.

5.4.4 Results on Deep Networks

Such a large reduction in evaluations during evolution over GLO allows TaylorGLO to tackle harder problems, including models that have millions of parameters. On the CIFAR-10, CIFAR-100, and SVHN datasets, TaylorGLO was able to outperform cross-entropy baselines consistently on a variety models, as shown in Table 5.1. These increases in accuracy are greater than what is possible through implicit learning rate adjustment alone (detailed in Section 5.4.5). TaylorGLO also provides further improvement on architectures

that use Cutout [35] or CutMix [197].

The models vary in structure, regularization techniques, and trainable parameter counts. For example, the AllCNN-C model has a basic, sequential layer structure and dropout regularization, while PyramidNet models have a branching structure with skip connections and batch normalization. Notably, these architectures were all manually designed and tuned against the cross-entropy loss. It is conceivable that small architectural modifications that are deleterious when training with the cross-entropy loss may be beneficial with TaylorGLO loss functions. This opens the door to the discovery of combinations of new architectures and loss functions that mutually benefit each other in the future.

It is also interesting that TaylorGLO improves upon architectures with several different regularization techniques already implemented. This result suggests that TaylorGLO's mechanism of avoiding overfitting is different from other regularization techniques. Thus, TaylorGLO can be a complementary form of regularization, and emphasize other regularization techniques' effects further.

5.4.5 Learning Rate Sensitivity

Loss functions can embody different learning rates implicitly. This section shows that TaylorGLO loss functions' benefits come from more than just metalearning such learning rates. Increases in performance that result from altering the base learning rate with cross-entropy loss are significantly

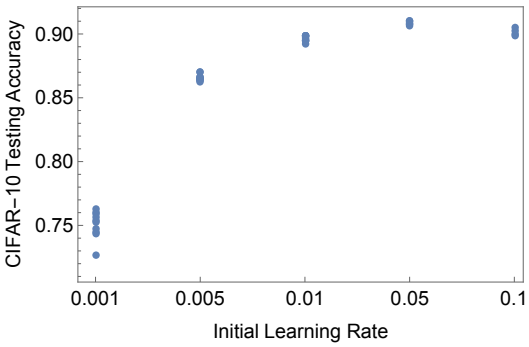


Figure 5.6: **Effect of varying learning rates in AllCNN-C when trained with the cross-entropy loss on CIFAR-10.** For each learning rate, ten models were trained, with up to ten retries if training failed. The majority of training attempts failed for learning rates larger than 0.01. The 0.01 learning rate used in the experiments in this chapter results in best stable performance. Overall, the small performance differences that can result from adjusting the learning rate, regardless of stability, are much smaller than those that result from training with TaylorGLO. Thus, TaylorGLO provides a mechanism for improvement beyond implicit adjustments of the learning rate.

smaller than those that TaylorGLO provides.

More specifically, Figure 5.6 quantifies the effect of varying learning rates on the final testing accuracy of AllCNN-C models trained on CIFAR-10. AllCNN-C was chosen for this analysis since it exhibits the largest variations in performance, making this effect more clear. While learning rates larger than 0.01 (the standard learning rate for AllCNN-C) reach slightly higher accuracies, this effect comes at the cost of less stable training. The majority of models trained with these higher learning rates failed to train. Thus, the standard choice of learning rate for AllCNN-C is appropriate for the cross-entropy loss, and TaylorGLO loss functions are able to improve upon it.

Table 5.2: **Performance of Taylor approximations of the cross-entropy loss function on AllCNN-C with CIFAR-10.** Approximations of different orders, with $\mathbf{a} = \langle 0.5, 0.5 \rangle$, are presented. Presented accuracies are the mean from ten runs. The baseline is the standard cross-entropy loss. Higher-order approximations are better, suggesting a potential (although computationally expensive) opportunity for improvement in the future.

Loss Function	Mean Accuracy (stddev)
$k = 2$	0.1034 (0.0101)
$k = 3$	0.8451 (0.0043)
$k = 4$	0.8592 (0.0032)
$k = 5$	0.8649 (0.0042)
Cross-Entropy	**0.8965 (0.0021)**

5.4.6 Comparison to Cross-Entropy Loss Taylor Approximations

While TaylorGLO's performance originates primarily from discovering better loss functions, it is informative to analyze what role the accuracy of the Taylor approximation plays in it. One way to characterize this effect is to analyze the performance of various Taylor approximations of the cross-entropy loss.

Table 5.2 provides results from such a study. Bivariate approximations to the cross-entropy loss, centered at $\mathbf{a} = \langle 0.5, 0.5 \rangle$, with different orders k were used to train AllCNN-C models on CIFAR-10. Third-order approximations and above are trainable. Approximations' performance is within a few percentage points of the cross-entropy loss, with higher-order approximations yielding progressively better accuracies, as expected.

The results thus show that third-order TaylorGLO loss functions can-

not represent the cross-entropy baseline loss accurately. One possibility for improving TaylorGLO is thus to utilize higher order approximations. However, it is remarkable that TaylorGLO can still find loss functions that outperform the cross-entropy loss. Also, the increase in the number of parameters—and the corresponding increase in computational requirements—may in practice outweigh the benefits from a finer-grained representation. This effect was seen in preliminary experiments, and the third-order approximations (used in this chapter) deemed to strike a good balance.

5.5 Experimental Analysis of TaylorGLO Models

The previous section evaluated TaylorGLO's performance on a variety of models and datasets, demonstrating that customized loss functions result in significant improvements in accuracy. This section analyzes the specific differences in models trained with TaylorGLO loss functions compared to those without.

5.5.1 Trained Model Surfaces

Not only does TaylorGLO train more accurate models, but its loss functions result in more robust models. Robustness is an important characteristic in networks. It is closely related to generalization: They both emerge in networks whose performance is not highly sensitive to the specific values of trained weights. Such models also tend to maintain their performance better following quantization, noise, loss of input or internal elements, etc.

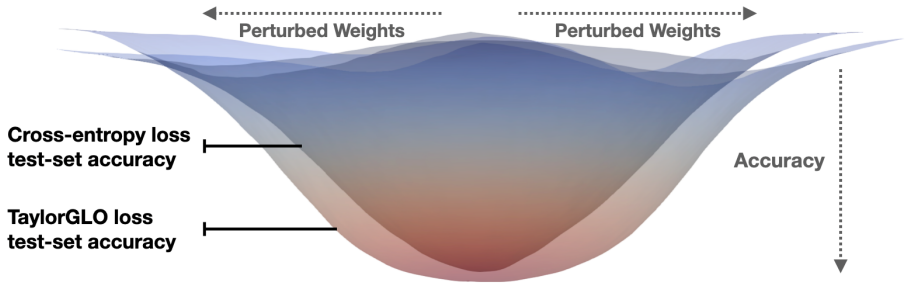


Figure 5.7: **Accuracy basins for AllCNN-C models trained with cross-entropy and TaylorGLO loss functions.** Accuracies are plotted along the vertical axis for perturbations along two random basis vectors on the horizontal axes. Higher accuracies are colored red. The TaylorGLO basin is both flatter and lower than that for the cross-entropy loss, indicating that small perturbations have a less negative impact on performance. Thus, networks trained with TaylorGLO loss functions are more robust and generalize better [90], which results in higher accuracy.

Robustness can be observed by evaluating a trained model and seeing how performance changes as weights are perturbed. In Figure 5.7, accuracy basins for two AllCNN-C models—one trained with the TaylorGLO loss function and another with the cross-entropy loss—are plotted along a two-dimensional slice $[-1, 1]$ of the weight space using a prior loss surface visualization technique [102]. At the core of the technique, two random unit vectors are chosen from the parameter space. The two vectors form a basis upon which a slice of the trainable parameter space is analyzed. These vectors are normalized in a filter-wise manner to accommodate network weights' scale invariance, thus ensuring that visualizations for two separate models can be compared. In sufficiently high-dimensional spaces, these vectors are guaranteed to be nearly orthogonal, thus the basis is nearly orthogonal. As a

result of the randomness, this parameter space slice is unbiased and should take all parameters into account to some degree. It can therefore be used to systematically perturb trainable parameters.

Using the random, filter-normalized basis, a two-dimensional grid can be formed, with the trained network at the center. Each non-center point in the grid represents a slightly perturbed variant of the trained network. Each of these networks is evaluated on the testing dataset, where accuracy is measured. These values on the grid form an accuracy basin. The fact that trained networks lie at network minima contributes to the basin shape.

The TaylorGLO loss function results in trained networks with flatter, lower basins. This result suggests that the model is more robust, i.e. its performance is less sensitive to small perturbations in the weight space, and it also generalizes better [90]. A particular manifestation of this robustness—that is, robustness against adversarial attacks—is anayzed in Section 7.3.

5.5.2 Biasing Optimization to a New Region

Optimization methods in general are biased in they are more likely to reach parameter vectors that lie in a distinct region of the parameter space. Ideally, this different region has better generalization properties. One way to observe this behavior empirically is to plot histograms of the weights. If the histograms noticeably differ, one can conclude that the compared solutions are in considerably different types of regions (the inverse, however, is not necessarily true).

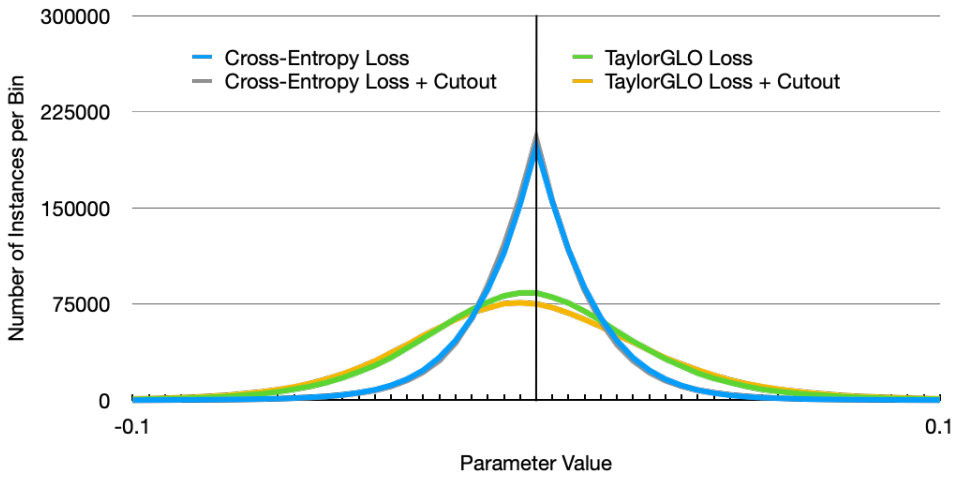


Figure 5.8: **Weight distributions for AllCNN-C models trained with cross-entropy and TaylorGLO loss functions on CIFAR-10.** The cross-entropy loss results in Laplace weight distributions, while TaylorGLO loss functions result in normally distributed weights. These different distributions show how TaylorGLO guides training towards a fundamentally different region of the weight space, which empirically results in better performance.

Table 5.3: **Weight characteristics for AllCNN-C models trained on CIFAR-10.** TaylorGLO results in models with higher L_2 weight norms than the cross-entropy loss, even though all configurations include weight decay during training. This finding suggests that smaller weights do not necessarily imply better generalization, as has been long believed [125, 191].

Training Configuration	L_2 Norm	Mean Value	Distribution
Cross-Entropy Loss	23.4278	0.0001	Laplace
Cross-Entropy Loss + Cutout	23.2174	0.0000	Laplace
TaylorGLO Loss	37.2540	-0.0019	Gaussian
TaylorGLO Loss + Cutout	40.8651	-0.0021	Gaussian

Figure 5.8 shows such a comparison of histograms trained under the cross-entropy loss and a TaylorGLO loss function. Each histogram has 51 bins to which each trainable parameter is assigned. Notably, weights from all networks trained with the cross-entropy loss follow a Laplace distribution, while those trained with a TaylorGLO loss function are normally distributed. These very different distributions show how TaylorGLO trains networks fundamentally differently from the cross-entropy loss. The addition of Cutout regularization does not significantly change the distribution of weights.

Table 5.3 presents weight statistics for each of the four configurations in Figure 5.8. Networks trained with TaylorGLO have significantly higher L_2 weight norms than those trained with the cross-entropy loss, even though they have the same level of weight decay. This observation concurs with past findings [48] that experimentally contradicted the long-held belief that networks with smaller weight norms generalize better [125, 191].

A mathematical justification for why different loss functions bias train-

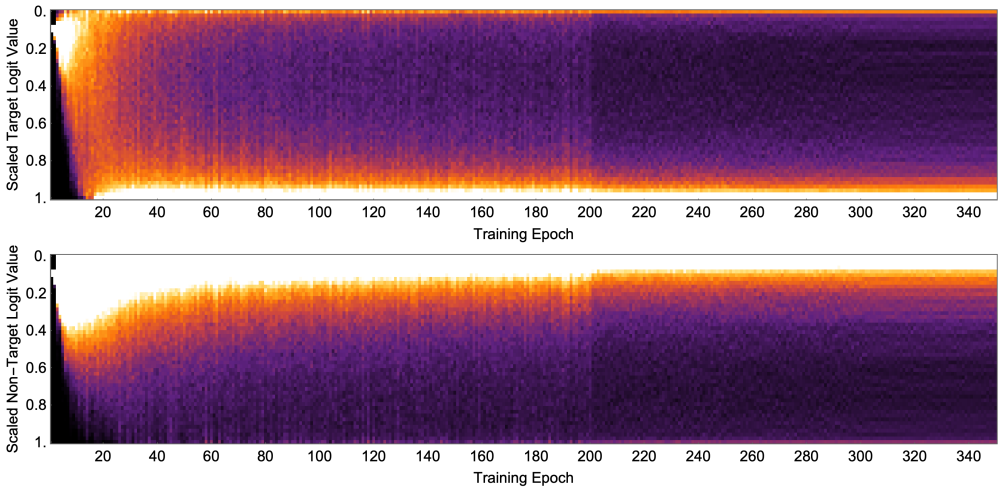
ing towards different regions of the weight space is provided in Chapter 6.

5.5.3 Loss Function Inputs Over Time

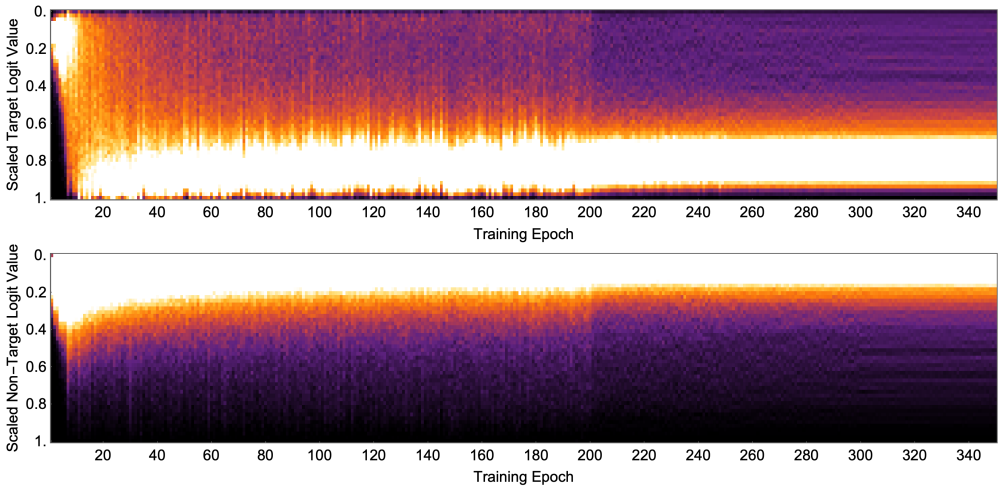
While TaylorGLO loss functions result in trained models that are different from those trained with the cross-entropy loss, the process by which they arive at these models is also substantially different. The training process can be empirically analyzed by observing how the distribution of a network's scaled logits changes over the training process.

Figure 5.9 presents scaled logit histograms for every epoch of training AllCNN-C networks on CIFAR-10 with both the cross-entropy loss and TaylorGLO. Histogram bin values are presented linearly through a color map, where "hotter" colors represent higher densities. These values are clipped to a maximum of 400 and 300 samples per bin for scaled target and non-target logits, respectively. This clipping shows more detail outside the main distribution modes since the vast majority of samples' logits are concentrated in relatively few bins. Clipped values are colored white in the scaled logit histograms. Overall, these histograms give insight into how model predictions for target and non-target labels change throughout the training process with different loss functions.

Since both TaylorGLO and cross-entropy models are initialized similarly, target and non-target logits start out near 0.1, since this is a ten-class classification problem, i.e. $1/n = 0.1$). This property can be seen as a concentration of values near the 0.1 level on the y-axis in the far left of each plot in



(a) Cross-Entropy Scaled Logits on AllCNN-C



(b) TaylorGLO Scaled Logits on AllCNN-C

Figure 5.9: **Target and non-target scaled logit histograms for TaylorGLO and cross-entropy loss functions on AllCNN-C with CIFAR-10.** More frequent logit values are represented by warmer, lighter colors. The two loss functions result in qualitatively different training dynamics. Namely the dense, white bands on the right side of (a)-top and (b)-top are centered along different logit values (the y-axis) and have different variances. The TaylorGLO band, where most target predictions lie, particularly, has a higher variance and is spaced farther from the histograms' bottom border, showing how TaylorGLO penalizes overly-confident predictions.

Figure 5.9. Both models learn quickly in the first few dozen epochs, and target and non-target logits spread away from each other as the network learns. That is, the average distance between target and non-target logits grows over time (from left to right). Over the duration of the training process, the white bands that contain most scaled logits increasingly contain more scaled logits, indicating the correct classification of more samples. The sharp transitions at 200 and 300 epochs are due to a scheduled learning rate decay. Smaller weight updates result in smoother distribution changes between adjacent epochs. This can be seen as a seeming reduction in noise along the x-axis in plots.

There are many visible differences between the cross-entropy loss and TaylorGLO. Most significantly, the main mode for scaled target logits on TaylorGLO (Figure 5.9 (b)-top) is wider—that is, the prominent white band is taller along the y-axis—than that for the cross-entropy loss (Figure 5.9 (a)-top) and is centered farther away from 1.0. This shows how TaylorGLO penalizes logits that are overly confident. On networks trained with TaylorGLO, predictions have more nuance (i.e., they have a larger vertical spread in the figures) and are less overtly categorical.

The cross-entropy loss also results in a second mode in the scaled target logit distributions (Figure 5.9 (a)-top). This mode, visible as a horizontal orange line near 0.0, indicates a population of misclassifications that is relatively consistent through the training process (i.e., left to right). These are misclassifications since target logit values less than 0.1 imply that all non-target logits would be larger than the target logit, since this is a ten-class domain. Notably,

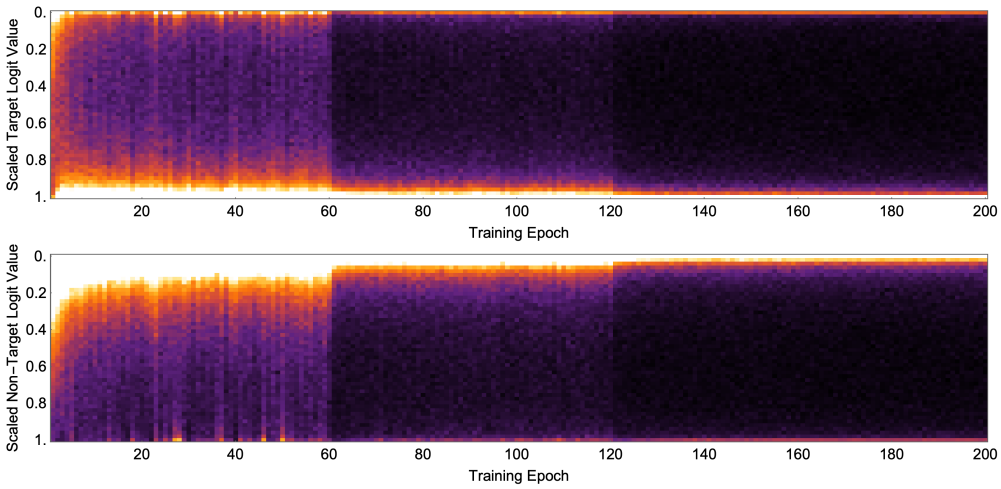
this band does not exist for TaylorGLO (Figure 5.9 (b)-top). On TaylorGLO, there is only a single major band in the latter parts of training (i.e., the right side of the Figure 5.9 (b)-top), and the values of the target logits in this band are greater than 0.1.

On the Wide ResNet 28-5 architecture (Figure 5.10), TaylorGLO results in generally similar behavior. As on AllCNN-C, the main target logit mode is wider and centered away from 1.0, i.e. away from the plots' bottom border. However, it is even more distanced than on AllCNN-C. These completely different training dynamics, demonstrate how TaylorGLO discovers different loss functions that are customized to individual architectures.

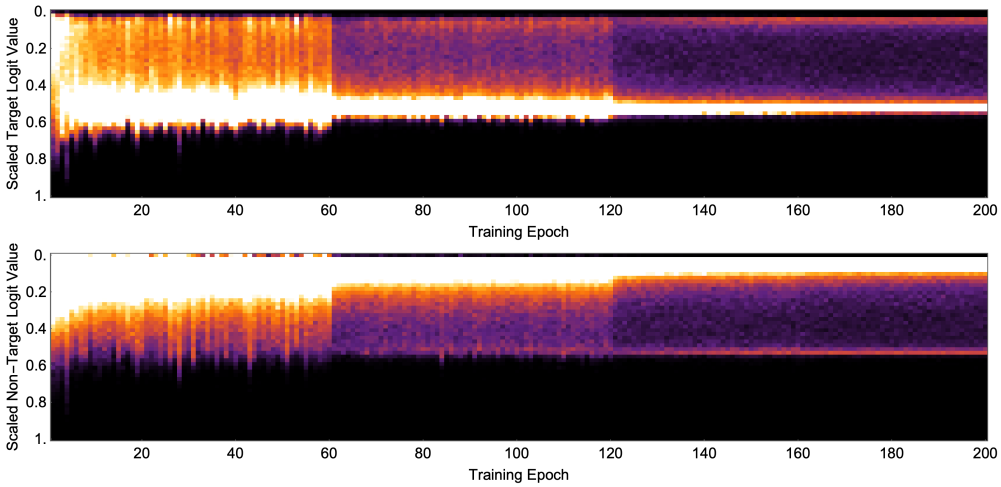
5.6 Discussion

TaylorGLO was shown to more efficiently and effectively find customized loss functions than GLO. The constrained search space and single-phase evolution that TaylorGLO use require many fewer time-consuming candidate evaluations. These efficiency improvements allow TaylorGLO to be applied to larger deep neural networks, where it finds loss functions that outperform the cross-entropy loss.

TaylorGLO achieves its performance gains by training networks whose trainable weights reach a different point in weight space than if the cross-entropy loss were used. These new solutions are shown to wider and flatter, a strong indicator of robustness and generalization. An analysis of the distribution of weights shows that TaylorGLO results in distinctly different trained



(a) Cross-Entropy Scaled Logits on Wide ResNet 28-5



(b) TaylorGLO Scaled Logits on Wide ResNet 28-5

Figure 5.10: **Target and non-target scaled logit histograms on Wide ResNet 28-5 with CIFAR-10.** Both loss functions result in qualitatively different training dynamics. Like on AllCNN-C, the dense, white bands on the right of (a)-top and (b)-top are centered along different logit values and have different entropy. However, unlike on AllCNN-C, the specific values and sizes of the bands in (b) differ. Thus, the TaylorGLO loss functions for AllCNN-C and Wide ResNet 28-5 have different training dynamics, indicating that TaylorGLO loss functions are customized to each architecture.

networks. These networks' training have completely different dynamics that further demonstrate how TaylorGLO can provide a type of regularization. However, what is still lacking is a comprehensive theoretical understanding that demonstrates why and how TaylorGLO loss functions have these effects. The next chapter provides such a framework.

The level of improvement on trained model performance that TaylorGLO loss functions are able to confer varies for different types of architectures, and even varying scales—depth and/or width—of a particular architecture. There is no clear relationship between TaylorGLO efficacy and general architecture morphologies. Even seemingly small changes—such as those between ResNet-20 and Pre ResNet-20—can have an impact on how much TaylorGLO can improve over the baseline. Many architecture characteristics affect the performance of TaylorGLO loss functions. Thus, understanding these effects and leveraging them is an important direction for future work.

While TaylorGLO outperforms the original GLO technique, GLO is far from obsolete. In cases where a loss function may have several different inputs, GLO scales far better than TaylorGLO, which would require a very large increase in the number of paramters (as described in Equation 5.4). There are many opportunities to leverage this flexibility in future work.

5.7 Conclusion

This chapter presented TaylorGLO, a new technique for loss-function metalearning that embodies a practical refinement of the core ideas behind

the GLO technique. TaylorGLO leverages a novel parameterization for loss functions, allowing the use of continuous optimization rather than genetic programming for the search, thus making it more efficient and more reliable. TaylorGLO loss functions serve to regularize the learning task, outperforming the standard cross-entropy loss significantly on MNIST, CIFAR-10, and SVHN benchmark tasks with a variety of network architectures. They also outperform loss functions discovered by GLO, while requiring many fewer candidates to be evaluated during search. Thus, TaylorGLO results in higher testing accuracies, better data utilization, and more robust models, and is a promising avenue for metalearning. Further analyses of the TaylorGLO technique show how TaylorGLO loss functions result in fundamentally different training processes that guide models towards quantitatively different parts of the trainable weight space with flatter minima. The underlying reasons for these differences are analyzed theoretically in Chapter 6.

Chapter 6

Understanding Regularization

Regularization is a key concept in deep learning: it guides learning towards configurations that are likely to perform robustly on unseen data. Different regularization approaches originate from intuitive understanding of the learning process and have been shown to be effective empirically. However, the understanding of the underlying mechanisms, the different types of regularization, and their interactions, is limited. Experiments in previous chapters suggest that metalearned loss functions serve as regularizers in a surprising but transparent way: they prevent the network from learning overly-confident predictions. While it may be too early to develop a comprehensive theory of deep network regularization, given the relatively nascent state of this area, it is possible to make progress in understanding regularization of this specific type, as is done in this chapter.

6.1 Overview

Since metalearned loss functions are customized to a given architecture-task pair, there needs to be a shared framework under which loss functions can be analyzed and compared. In the framework developed in this chapter, the

stochastic gradient descent (SGD) learning rule is decomposed to coefficient expressions that can be defined for a wide range of loss functions. These expressions provide an intuitive understanding of the training dynamics in specific contexts.

Using this framework, mean squared error (MSE), cross-entropy, Baikal, and third-order TaylorGLO loss functions are analyzed at the null epoch, when network weights are similarly distributed, and in a zero training error regime, where the training samples' labels have been perfectly memorized. These scenarios show the implicit biases that different loss functions impart. For any intermediate point in the training process, the strength of the zero training error regime as an attractor is analyzed and a constraint on this property is derived on TaylorGLO parameters by characterizing how the output distribution's entropy changes. In a concrete TaylorGLO loss function that has been metalearned, these attraction dynamics are calculated for individual samples at every epoch in a real training run, and contrasted with those for the cross-entropy loss. This comparison provides clarity on how TaylorGLO avoids becoming overly confident in its predictions. Further, the analysis shows how label smoothing [167], a traditional type of regularization, can be implicitly encoded by TaylorGLO loss functions: Any representable loss function has label-smoothed variants that are also representable by the parameterization, meaning that TaylorGLO is able to take advantage of it and automatically learn optimal levels of label smoothing that synergize with other forms of regularization.

6.2 Learning Rule Decomposition

This section develops the framework for the analysis in this chapter. By decomposing the learning rules under different loss functions, comparisons can be drawn at different stages of the training process. Consider the standard SGD update rule:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \left(\mathcal{L}(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}) \right), \tag{6.1}$$

where η is the learning rate, $\mathcal{L}(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta})$ is the loss function applied to the network $h(\boldsymbol{x}_i, \boldsymbol{\theta})$, $\boldsymbol{x}_i$ is an input data sample, $\boldsymbol{y}_i$ is the ith sample's corresponding label, and $\boldsymbol{\theta}$ is the set of trainable parameters in the model. The update for a single weight θ_j is

$$\theta_j \leftarrow \theta_j - \eta D_{\boldsymbol{j}} \left(\mathcal{L}(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}) \right) = \theta_j - \eta \frac{\partial}{\partial s} \mathcal{L}(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta} + s\boldsymbol{j}) \Big|_{s \to 0}, \tag{6.2}$$

where $\boldsymbol{j}$ is a basis vector for the jth weight.

The remainder of this section illustrates decompositions of this general learning rule in a classification context for a variety of loss functions: mean squared error (MSE), the cross-entropy loss function, the general third-order TaylorGLO loss function, and the Baikal loss function. Each decomposition results in a learning rule of the form

$$\theta_j \leftarrow \theta_j + \eta \frac{1}{n} \sum_{k=1}^{n} \left[\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}) D_{\boldsymbol{j}} \left(h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) \right) \right], \tag{6.3}$$

where $\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta})$ is an expression that is specific to each loss function. This expression dictates how individual logits' gradients affect weight updates. Specific instance of $\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta})$ can thus be used to analyze and compare the be-

havior of their corresponding loss functions, as will be shown in the following sections.

Substituting the **Mean squared error (MSE)** loss into Equation 6.2,

$$\theta_j \leftarrow \theta_j - \eta \frac{1}{n} \sum_{k=1}^{n} \left[2 \left(h_k(\boldsymbol{x}_i, \boldsymbol{\theta} + s\boldsymbol{j}) - y_{ik} \right) \frac{\partial}{\partial s} h_k(\boldsymbol{x}_i, \boldsymbol{\theta} + s\boldsymbol{j}) \right] \Big|_{s \to 0} \tag{6.4}$$

$$= \theta_j + \eta \frac{1}{n} \sum_{k=1}^{n} \left[2 \left(y_{ik} - h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) \right) \frac{\partial}{\partial s} h_k(\boldsymbol{x}_i, \boldsymbol{\theta} + s\boldsymbol{j}) \Big|_{s \to 0} \right], \tag{6.5}$$

and breaking up the coefficient expressions into $\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta})$ results in the weight update step

$$\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}) = 2y_{ik} - 2h_k(\boldsymbol{x}_i, \boldsymbol{\theta}). \tag{6.6}$$

Substituting the **Cross-entropy loss** into Equation 6.2,

$$\theta_j \leftarrow \theta_j + \eta \frac{1}{n} \sum_{k=1}^{n} \left[y_{ik} \frac{1}{h_k(\boldsymbol{x}_i, \boldsymbol{\theta} + s\boldsymbol{j})} \frac{\partial}{\partial s} h_k(\boldsymbol{x}_i, \boldsymbol{\theta} + s\boldsymbol{j}) \right] \Big|_{s \to 0} \tag{6.7}$$

$$= \theta_j + \eta \frac{1}{n} \sum_{k=1}^{n} \left[\frac{y_{ik}}{h_k(\boldsymbol{x}_i, \boldsymbol{\theta})} \frac{\partial}{\partial s} h_k(\boldsymbol{x}_i, \boldsymbol{\theta} + s\boldsymbol{j}) \Big|_{s \to 0} \right], \tag{6.8}$$

and breaking up the coefficient expressions into $\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta})$ results in the weight update step

$$\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}) = \frac{y_{ik}}{h_k(\boldsymbol{x}_i, \boldsymbol{\theta})}. \tag{6.9}$$

Substituting the **Baikal loss** into Equation 6.2,

$$\theta_j \leftarrow \theta_j + \eta \frac{1}{n} \sum_{k=1}^{n} \left[\left(\frac{1}{h_k(\boldsymbol{x}_i, \boldsymbol{\theta} + s\boldsymbol{j})} + \frac{y_{ik}}{h_k(\boldsymbol{x}_i, \boldsymbol{\theta} + s\boldsymbol{j})^2} \right) \frac{\partial}{\partial s} h_k(\boldsymbol{x}_i, \boldsymbol{\theta} + s\boldsymbol{j}) \right] \Big|_{s \to 0} \tag{6.10}$$

$$= \theta_j + \eta \frac{1}{n} \sum_{k=1}^{n} \left[\left(\frac{1}{h_k(\boldsymbol{x}_i, \boldsymbol{\theta})} + \frac{y_{ik}}{h_k(\boldsymbol{x}_i, \boldsymbol{\theta})^2} \right) \frac{\partial}{\partial s} h_k(\boldsymbol{x}_i, \boldsymbol{\theta} + s\boldsymbol{j}) \Big|_{s \to 0} \right], \tag{6.11}$$

and breaking up the coefficient expressions into $\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta})$ results in the weight update step

$$\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}) = \frac{1}{h_k(\boldsymbol{x}_i, \boldsymbol{\theta})} + \frac{y_{ik}}{h_k(\boldsymbol{x}_i, \boldsymbol{\theta})^2}. \tag{6.12}$$

Substituting the **Third-order TaylorGLO loss** with parameters $\boldsymbol{\lambda}$ into Equation 6.2,

$$\begin{aligned}
\theta_j \leftarrow \theta_j + \eta \frac{1}{n} \sum_{k=1}^{n} \Bigg[\\
\lambda_2 \frac{\partial}{\partial s} h_k(\boldsymbol{x}_i, \boldsymbol{\theta} + s\boldsymbol{j}) + \lambda_3 2 \left(h_k(\boldsymbol{x}_i, \boldsymbol{\theta} + s\boldsymbol{j}) - \lambda_1\right) \frac{\partial}{\partial s} h_k(\boldsymbol{x}_i, \boldsymbol{\theta} + s\boldsymbol{j}) \\
+\lambda_4 3 \left(h_k(\boldsymbol{x}_i, \boldsymbol{\theta} + s\boldsymbol{j}) - \lambda_1\right)^2 \frac{\partial}{\partial s} h_k(\boldsymbol{x}_i, \boldsymbol{\theta} + s\boldsymbol{j}) + \lambda_5 (y_{ik} - \lambda_0) \frac{\partial}{\partial s} h_k(\boldsymbol{x}_i, \boldsymbol{\theta} + s\boldsymbol{j}) \\
+ \left(\lambda_6 (y_{ik} - \lambda_0) 2 \left(h_k(\boldsymbol{x}_i, \boldsymbol{\theta} + s\boldsymbol{j}) - \lambda_1\right) + \lambda_7 (y_{ik} - \lambda_0)^2\right) \frac{\partial}{\partial s} h_k(\boldsymbol{x}_i, \boldsymbol{\theta} + s\boldsymbol{j}) \Bigg] \Bigg|_{s \to 0}
\end{aligned} \tag{6.13}$$

$$\begin{aligned}
= \theta_j + \eta \frac{1}{n} \sum_{k=1}^{n} \Bigg[\left(\lambda_3 + \lambda_6 (y_{ik} - \lambda_0)\right) 2 \left(h_k(\boldsymbol{x}_i, \boldsymbol{\theta} + s\boldsymbol{j}) - \lambda_1\right) \frac{\partial}{\partial s} h_k(\boldsymbol{x}_i, \boldsymbol{\theta} + s\boldsymbol{j}) \Big|_{s \to 0} \\
+ \left(\lambda_2 + \lambda_5 (y_{ik} - \lambda_0) + \lambda_7 (y_{ik} - \lambda_0)^2\right) \frac{\partial}{\partial s} h_k(\boldsymbol{x}_i, \boldsymbol{\theta} + s\boldsymbol{j}) \Big|_{s \to 0} \\
+\lambda_4 3 \left(h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) - \lambda_1\right)^2 \frac{\partial}{\partial s} h_k(\boldsymbol{x}_i, \boldsymbol{\theta} + s\boldsymbol{j}) \Big|_{s \to 0} \Bigg],
\end{aligned} \tag{6.14}$$

and breaking up the coefficient expressions into $\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta})$ results in the weight update step

$$\begin{aligned}
\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}) = \left(\lambda_3 + \lambda_6 (y_{ik} - \lambda_0)\right) 2 \left(h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) - \lambda_1\right) \\
+\lambda_2 + \lambda_5 (y_{ik} - \lambda_0) + \lambda_7 (y_{ik} - \lambda_0)^2 + \lambda_4 3 \left(h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) - \lambda_1\right)^2
\end{aligned} \tag{6.15}$$

$$\begin{aligned}
= 2\lambda_3 h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) - 2\lambda_1 \lambda_3 + 2\lambda_6 h_k(\boldsymbol{x}_i, \boldsymbol{\theta})(y_{ik} - \lambda_0) - 2\lambda_1 \lambda_6 (y_{ik} - \lambda_0) + \lambda_2 + \lambda_5 y_{ik} \\
-\lambda_5 \lambda_0 + \lambda_7 y_{ik}^2 - 2\lambda_7 \lambda_0 y_{ik} + \lambda_7 \lambda_0^2 + 3\lambda_4 h_k(\boldsymbol{x}_i, \boldsymbol{\theta})^2 - 6\lambda_1 \lambda_4 h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) + 3\lambda_4 \lambda_1^2
\end{aligned} \tag{6.16}$$

$$= 2\lambda_3 h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) - 2\lambda_1\lambda_3 + 2\lambda_6 h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) y_{ik} - 2\lambda_6\lambda_0 h_k(\boldsymbol{x}_i, \boldsymbol{\theta})$$
$$-2\lambda_1\lambda_6 y_{ik} + 2\lambda_1\lambda_6\lambda_0 + \lambda_2 + \lambda_5 y_{ik} - \lambda_5\lambda_0 + \lambda_7 y_{ik}^2 - 2\lambda_7\lambda_0 y_{ik} \tag{6.17}$$
$$+\lambda_7\lambda_0^2 + 3\lambda_4 h_k(\boldsymbol{x}_i, \boldsymbol{\theta})^2 - 6\lambda_1\lambda_4 h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) + 3\lambda_4\lambda_1^2.$$

To simplify analysis, $\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta})$ can be decomposed into a linear combination of $\langle 1, h_k(\boldsymbol{x}_i, \boldsymbol{\theta}), h_k(\boldsymbol{x}_i, \boldsymbol{\theta})^2, h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) y_{ik}, y_{ik}, y_{ik}^2 \rangle$ with respective coefficients $\langle c_1, c_h, c_{hh}, c_{hy}, c_y, c_{yy} \rangle$ whose values are implicitly functions of $\boldsymbol{\lambda}$:

$$\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}) = c_1 + c_h h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) + c_{hh} h_k(\boldsymbol{x}_i, \boldsymbol{\theta})^2 + c_{hy} h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) y_{ik} + c_y y_{ik} + c_{yy} y_{ik}^2 \tag{6.18}$$

where

$$c_1 = -2\lambda_1\lambda_3 + 2\lambda_0\lambda_1\lambda_6 + \lambda_2 - \lambda_5\lambda_0 + \lambda_7\lambda_0^2 + 3\lambda_4\lambda_1^2 \tag{6.19}$$
$$c_h = 2\lambda_3 - 2\lambda_6\lambda_0 - 6\lambda_4\lambda_1 \tag{6.20}$$
$$c_{hh} = 3\lambda_4 \tag{6.21}$$
$$c_{hy} = 2\lambda_6 \tag{6.22}$$
$$c_y = -2\lambda_1\lambda_6 + \lambda_5 - 2\lambda_7\lambda_0 \tag{6.23}$$
$$c_{yy} = \lambda_7. \tag{6.24}$$

This linear combination abstracts away complexity and results in simpler math when analyzing TaylorGLO.

Using the decomposition framework above, it is possible to characterize and compare training dynamics under different loss functions. In Section 6.3, the decompositions are first analyzed under a zero training error regime to identify optimization biases that lead to implicit regularization. In Section 6.4,

the opposite end of the training process is then analyzed, i.e. the null epoch. In Section 6.6, generalizing to the entire training process, a theoretical constraint is derived on the entropy of a network's outputs. Combined with experimental data, this constraint characterizes the data fitting and regularization processes that result from TaylorGLO training.

6.3 Zero Training Error Optimization Biases

Certain biases in optimization imposed by a loss function can be best observed in the case where there is nothing new to learn from the training data. Consider the case where there is zero training error, that is, $h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) - y_{ik} = 0$. In this case, all $h_k(\boldsymbol{x}_i, \boldsymbol{\theta})$ can be substituted with y_{ik} in $\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta})$, as is done below for the different loss functions.

6.3.1 Mean Squared Error (MSE)

In this case,

$$\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}) = 2y_{ik} - 2h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) = 0. \tag{6.25}$$

Thus, there are no changes to the weights of the model once error reaches zero. This observation contrasts with the earlier findings that discovered an implicit regularization effect when training with MSE loss *and* label noise [19]. Notably, this null behavior is representable in a non-degenerate TaylorGLO parameterization, since MSE is itself representable by TaylorGLO with $\boldsymbol{\lambda} = \langle 0, 0, 0, -1, 0, 2, 0, 0 \rangle$. Thus, this behavior can be leveraged in evolved loss functions.

6.3.2 Cross-Entropy Loss

Since $h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) = 0$ for non-target logits in a zero training error regime, $\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}) = \frac{0}{0}$, i.e. an indeterminate form. Thus, an arbitrarily-close-to-zero training error regime is analyzed instead, such that $h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) = \epsilon$ for non-target logits for an arbitrarily small ϵ. Since all scaled logits sum to 1, $h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) = 1 - (n-1)\epsilon$ for the target logit. Let us analyze the learning rule as ϵ tends towards 0:

$$\theta_j \leftarrow \theta_j + \lim_{\epsilon \to 0} \eta \frac{1}{n} \sum_{k=1}^{n} \begin{cases} \frac{y_{ik}}{\epsilon} D_{\boldsymbol{j}}\left(h_k(\boldsymbol{x}_i, \boldsymbol{\theta})\right) & y_{ik} = 0 \\ \frac{y_{ik}}{1-(n-1)\epsilon} D_{\boldsymbol{j}}\left(h_k(\boldsymbol{x}_i, \boldsymbol{\theta})\right) & y_{ik} = 1 \end{cases} \tag{6.26}$$

$$= \theta_j + \eta \frac{1}{n} \sum_{k=1}^{n} \begin{cases} 0 & y_{ik} = 0 \\ D_{\boldsymbol{j}}\left(h_k(\boldsymbol{x}_i, \boldsymbol{\theta})\right) & y_{ik} = 1. \end{cases} \tag{6.27}$$

Intuitively, this learning rule aims to increase the value of the target scaled logits. Since logits are scaled by a softmax function, increasing the value of one logit decreases the values of other logits. Thus, the fixed point of this bias will be to force non-target scaled logits to zero, and target scaled logits to one. In other words, this behavior aims to minimize the divergence between the predicted distribution and the training data's distribution.

TaylorGLO can represent this behavior, and can thus be leveraged in evolved loss functions, through any case where $a = 0$ and $b + c > 0$. Any $\boldsymbol{\lambda}$ where $\lambda_2 = 2\lambda_1\lambda_3 + \lambda_5\lambda_0 - 2\lambda_1\lambda_6\lambda_0 - \lambda_7\lambda_0^2 - 3\lambda_4\lambda_1^2$ represents such a satisfying family of cases. Additionally, TaylorGLO allows for the strength of this bias to be tuned independently from η by adjusting the magnitude of $b + c$.

6.3.3 Baikal Loss

Notably, the Baikal loss function results in infinite gradients at zero training error, rendering it unstable, even if using it to fine-tune from a previously trained network that already reached zero training error. However, the zero-error regime is irrelevant with Baikal because it cannot be reached in practice:

Theorem 6.3.1. *Zero training error regions of the weight space are not attractors for the Baikal loss function.*

The reason is that if a network reaches a training error that is arbitrarily close to zero, there is a repulsive effect that biases the model's weights away from zero training error.

Proof. Given that Baikal does tend to minimize training error to a large degree—otherwise it would be useless as a loss function since training data is assumed to be in-distribution— what happens as the learning rule approaches a point in parameter space that is arbitrarily-close to zero training error can be observed. Assume, without loss of generality, that all non-target scaled logits have the same value. Then,

$$\theta_j \leftarrow \theta_j + \eta \frac{1}{n} \sum_{k=1}^{n} \begin{cases} \lim\limits_{h_k(\boldsymbol{x}_i,\boldsymbol{\theta}) \to \frac{\epsilon}{n-1}} \gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}) D_{\boldsymbol{j}} \left(h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) \right) & y_{ik} = 0 \\ \lim\limits_{h_k(\boldsymbol{x}_i,\boldsymbol{\theta}) \to 1-\epsilon} \gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}) D_{\boldsymbol{j}} \left(h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) \right) & y_{ik} = 1 \end{cases} \tag{6.28}$$

$$= \theta_j + \eta \frac{1}{n} \sum_{k=1}^{n} \begin{cases} \lim\limits_{h_k(\boldsymbol{x}_i,\boldsymbol{\theta}) \to \frac{\epsilon}{n-1}} \left(\dfrac{1}{h_k(\boldsymbol{x}_i,\boldsymbol{\theta})} + \dfrac{0}{h_k(\boldsymbol{x}_i,\boldsymbol{\theta})^2} \right) D_{\boldsymbol{j}} \left(h_k(\boldsymbol{x}_i,\boldsymbol{\theta}) \right) & y_{ik} = 0 \\ \lim\limits_{h_k(\boldsymbol{x}_i,\boldsymbol{\theta}) \to 1-\epsilon} \left(\dfrac{1}{h_k(\boldsymbol{x}_i,\boldsymbol{\theta})} + \dfrac{1}{h_k(\boldsymbol{x}_i,\boldsymbol{\theta})^2} \right) D_{\boldsymbol{j}} \left(h_k(\boldsymbol{x}_i,\boldsymbol{\theta}) \right) & y_{ik} = 1 \end{cases} \tag{6.29}$$

$$= \theta_j + \eta \frac{1}{n} \sum_{k=1}^{n} \begin{cases} \dfrac{n-1}{\epsilon} D_{\boldsymbol{j}} \left(h_k(\boldsymbol{x}_i,\boldsymbol{\theta}) \right) & y_{ik} = 0 \\ \left(\dfrac{1}{1-\epsilon} + \dfrac{1}{(1-\epsilon)^2} \right) D_{\boldsymbol{j}} \left(h_k(\boldsymbol{x}_i,\boldsymbol{\theta}) \right) & y_{ik} = 1 \end{cases} \tag{6.30}$$

$$= \theta_j + \eta \frac{1}{n} \sum_{k=1}^{n} \begin{cases} \dfrac{n-1}{\epsilon} D_{\boldsymbol{j}} \left(h_k(\boldsymbol{x}_i,\boldsymbol{\theta}) \right) & y_{ik} = 0 \\ \dfrac{2-\epsilon}{\epsilon^2 - 2\epsilon + 1} D_{\boldsymbol{j}} \left(h_k(\boldsymbol{x}_i,\boldsymbol{\theta}) \right) & y_{ik} = 1 \, . \end{cases} \tag{6.31}$$

The behavior in the $y_{ik} = 0$ case will dominate for small values of ϵ. Both cases have a positive range for small values of ϵ, ultimately resulting in non-target scaled logits becoming maximized, and subsequently the non-target logit becoming minimized. This assertion is equivalent, in expectation, to saying that ϵ will become larger after applying the learning rule. A larger ϵ implies a move away from a zero training error area of the parameter space. Thus, zero training error is not an attractor for the Baikal loss function. □

6.3.4 Third-Order TaylorGLO Loss

According to Equation 6.18, in the zero-error regime $\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta})$ can be written as a linear combination of $\langle 1, y_{ik}, y_{ik}^2 \rangle$ and $\langle a, b, c \rangle$:

$$\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}) = a + b y_{ik} + c y_{ik}^2, \tag{6.32}$$

where

$$a = \lambda_2 - 2\lambda_1\lambda_3 - \lambda_5\lambda_0 + 2\lambda_1\lambda_6\lambda_0 + \lambda_7\lambda_0^2 + 3\lambda_4\lambda_1^2 \tag{6.33}$$

$$b = 2\lambda_3 - 2\lambda_6\lambda_0 - 2\lambda_1\lambda_6 + \lambda_5 - 2\lambda_7\lambda_0 - 6\lambda_4\lambda_1 \tag{6.34}$$

$$c = 2\lambda_6 + \lambda_7 + 3\lambda_4. \tag{6.35}$$

Notably, in the basic classification case, $\forall w \in \mathbb{N}_1 : y_{ik} = y_{ik}^w$, since $y_{ik} \in \{0, 1\}$. This observation provides an intuition for why higher-order TaylorGLO loss functions are not able to provide fundamentally different behavior (beyond a more overparameterized search space), and thus no improvements in performance over third-order loss functions. The learning rule thus becomes

$$\theta_j \leftarrow \theta_j + \eta \frac{1}{n} \sum_{k=1}^{n} \begin{cases} aD_{\boldsymbol{j}}\left(h_k(\boldsymbol{x}_i, \boldsymbol{\theta})\right) & y_{ik} = 0 \\ (a+b+c)D_{\boldsymbol{j}}\left(h_k(\boldsymbol{x}_i, \boldsymbol{\theta})\right) & y_{ik} = 1. \end{cases} \tag{6.36}$$

As a concrete example, consider the loss function TaylorGLO discovered for the AllCNN-C model on CIFAR-10. It had $a = -373.917$, $b = -129.928$, $c = -11.3145$. Notably, all three coefficients are negative, i.e. all changes to θ_j are negatively scaled values of $D_{\boldsymbol{j}}\left(h_k(\boldsymbol{x}_i, \boldsymbol{\theta})\right)$, as can be seen from Equation 6.36. Thus, there are two competing processes in this learning rule: one that aims to minimize all non-target scaled logits (increasing the scaled logit distribution's entropy), and one that aims to minimize the target scaled logit (decreasing the scaled logit distribution's entropy). The processes conflict with each other since logits are scaled through a softmax function. These processes can shift weights in a particular way while maintaining zero training error, which results in implicit regularization. If, however, such shifts in this zero training error

regime do lead to misclassifications on the training data, $h_k(\boldsymbol{x}_i, \boldsymbol{\theta})$ would no longer equal y_{ik}, and a non-zero error regime's learning rule would come into effect. It would strive to get back to zero training error with a different $\boldsymbol{\theta}$.

Similarly to Baikal loss, a training error of exactly zero is not an attractor for some third-order TaylorGLO loss functions (this property can be seen through an analysis similar to that of Theorem 6.3.1). The zero-error case would occur in practice only if this loss function were to be used to fine tune a network that truly has a zero training error. It is, however, a useful step in characterizing the behavior of TaylorGLO, as will be seen in Section 6.6.

6.4 Behavior at the Null Epoch

Consider the null epoch, i.e., the first epoch of training. Assume all weights are randomly initialized:

$$\forall k \in [1, n], \text{where } n \geq 2 : \mathop{\mathbb{E}}_{i}[h_k(\boldsymbol{x}_i, \boldsymbol{\theta})] = \frac{1}{n}. \tag{6.37}$$

That is, logits are distributed with high entropy. Behavior at the null epoch can then be defined piecewise for target vs. non-target logits for each loss function.

In the case of **Mean squared error (MSE),**

$$\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}) = \begin{cases} 2n^{-1} & y_{ik} = 0 \\ 2n^{-1} - 2 & y_{ik} = 1. \end{cases} \tag{6.38}$$

Since $n \geq 2$, the $y_{ik} = 1$ case will always be negative, while the $y_{ik} = 0$ case will always be positive. Thus, target scaled logits will be maximized and non-target scaled logits minimized.

In the case of **Cross-entropy loss,**

$$\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}) = \begin{cases} 0 & y_{ik} = 0 \\ n & y_{ik} = 1. \end{cases} \tag{6.39}$$

Target scaled logits are maximized and, consequently, non-target scaled logits minimized as a result of the softmax function.

Similarly in the case of **Baikal loss,**

$$\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}) = \begin{cases} n & y_{ik} = 0 \\ n + n^2 & y_{ik} = 1. \end{cases} \tag{6.40}$$

Target scaled logits are minimized and, consequently, non-target scaled logits minimized as a result of the softmax function (since the $y_{ik} = 1$ case dominates).

In the case of **Third-order TaylorGLO loss,** since behavior is highly dependent on $\boldsymbol{\lambda}$, consider the concrete loss function used above:

$$\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}) = \begin{cases} -373.9170 - 130.2640\ h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) \\ \qquad -11.2188\ h_k(\boldsymbol{x}_i, \boldsymbol{\theta})^2 & y_{ik} = 0 \\ -372.4707 - 131.4700\ h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) \\ \qquad -11.2188\ h_k(\boldsymbol{x}_i, \boldsymbol{\theta})^2 & y_{ik} = 1. \end{cases} \tag{6.41}$$

Note that Equation 6.36 is a special case of this behavior where $h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) = y_{ik}$. Let us substitute $h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) = \frac{1}{n}$ (i.e., the expected value of a logit at the null epoch):

$$\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}) = \begin{cases} -373.9170 - 130.2640\ n^{-1} - 11.2188\ n^{-2} & y_{ik} = 0 \\ -372.4707 - 131.4700\ n^{-1} - 11.2188\ n^{-2} & y_{ik} = 1. \end{cases} \tag{6.42}$$

Since this loss function was found on CIFAR-10, a 10-class image classification task, $n = 10$:

$$\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}) = \begin{cases} -386.9546 & y_{ik} = 0 \\ -385.7299 & y_{ik} = 1. \end{cases} \tag{6.43}$$

Since both cases of $\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta})$ are negative, this behavior implies that all scaled logits will be minimized. However, since the scaled logits are the output of a softmax function, and the $y_{ik} = 0$ case is more strongly negative, the non-target scaled logits will be minimized more than the target scaled logits, resulting in a maximization of the target scaled logits.

The desired behavior at the null epoch is clear, and the above evaluated loss functions all exhibit it. However, certain settings for $\boldsymbol{\lambda}$ in TaylorGLO loss functions may have wholly behavior that is detrimental. Thus, a constraint on $\boldsymbol{\lambda}$ can be derived. Such a constraint is derived in Section 6.5 and used to improve the TaylorGLO search process in Section 7.2.

6.5 TaylorGLO Parameters at the Null Epoch

There are many different instances of $\boldsymbol{\lambda}$ for which models are untrainable. One such case, albeit a degenerate one, is $\boldsymbol{\lambda} = \mathbf{0}$, i.e. a function with zero gradients everywhere. Given the training dynamics at the null epoch (characterized in Section 6.4), more general constraints on $\boldsymbol{\lambda}$

Theorem 6.5.1. *A third-order TaylorGLO loss function is not trainable if the following constraints on $\boldsymbol{\lambda}$ are satisfied:*

$$c_1 + c_y + c_{yy} + \frac{c_h + c_{hy}}{n} + \frac{c_{hh}}{n^2} < (n-1)\left(c_1 + \frac{c_h}{n} + \frac{c_{hh}}{n^2}\right) \quad (6.44)$$

$$c_y + c_{yy} + \frac{c_{hy}}{n} < (n-2)\left(c_1 + \frac{c_h}{n} + \frac{c_{hh}}{n^2}\right). \quad (6.45)$$

Proof. At the null epoch, a valid loss function aims to, in expectation, minimize non-target scaled logits while maximizing target scaled logits. Thus, specific

cases can of $\boldsymbol{\lambda}$ can be found for which these behaviors occur. Considering the representation for $\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta})$ in Equation 6.18:

$$\theta_j \leftarrow \theta_j + \eta \frac{1}{n} \sum_{k=1}^{n} \begin{cases} \left(c_1 + c_h h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) + c_{hh} h_k(\boldsymbol{x}_i, \boldsymbol{\theta})^2\right) D_{\boldsymbol{j}} \left(h_k(\boldsymbol{x}_i, \boldsymbol{\theta})\right) & y_{ik} = 0 \\ \big(c_1 + c_h h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) + c_{hh} h_k(\boldsymbol{x}_i, \boldsymbol{\theta})^2 \\ \quad + c_{hy} h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) + c_y + c_{yy}\big) D_{\boldsymbol{j}} \left(h_k(\boldsymbol{x}_i, \boldsymbol{\theta})\right) & y_{ik} = 1 \end{cases}. \tag{6.46}$$

Substituting $h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) = \frac{1}{n}$ (i.e., the expected value of a logit at the null epoch),

$$\theta_j \leftarrow \theta_j + \eta \frac{1}{n} \sum_{k=1}^{n} \begin{cases} \left(c_1 + \frac{c_h}{n} + \frac{c_{hh}}{n^2}\right) D_{\boldsymbol{j}} \left(h_k(\boldsymbol{x}_i, \boldsymbol{\theta})\right) & y_{ik} = 0 \\ \left(c_1 + c_y + c_{yy} + \frac{c_h + c_{hy}}{n} + \frac{c_{hh}}{n^2}\right) D_{\boldsymbol{j}} \left(h_k(\boldsymbol{x}_i, \boldsymbol{\theta})\right) & y_{ik} = 1 \end{cases}. \tag{6.47}$$

For the degenerate behavior to appear, the directional derivative's coefficient in the $y_{ik} = 1$ case must be less than zero:

$$c_1 + c_y + c_{yy} + \frac{c_h + c_{hy}}{n} + \frac{c_{hh}}{n^2} \quad < \quad 0. \tag{6.48}$$

This finding can be made more general by asserting that the directional derivative's coefficient in the $y_{ik} = 1$ case be less than $(n-1)$ times the coefficient in the $y_{ik} = 0$ case, thus arriving at the following constraint on $\boldsymbol{\lambda}$:

$$c_1 + c_y + c_{yy} + \frac{c_h + c_{hy}}{n} + \frac{c_{hh}}{n^2} \quad < \quad (n-1)\left(c_1 + \frac{c_h}{n} + \frac{c_{hh}}{n^2}\right) \tag{6.49}$$

$$c_y + c_{yy} + \frac{c_{hy}}{n} \quad < \quad (n-2)\left(c_1 + \frac{c_h}{n} + \frac{c_{hh}}{n^2}\right) \tag{6.50}$$

□

These constraints restrict loss functions to pass through the regions at $\epsilon = 1 - \frac{1}{n}$ in Figure 6.3 where the entropy is increasing (colored red). The

inverse of these constraints may be used as invariants during loss function evolution, as will be described in Section 7.2.

6.6 Data Fitting and Regularization Processes

Under what gradient conditions does a network's softmax function transition from increasing the entropy in the output distribution to decreasing it? Let us analyze the case where all non-target logits have the same value, $\frac{\epsilon}{n-1}$, and the target logit has the value $1-\epsilon$. That is, all non-target classes have equal probabilities.

6.6.1 Softmax Entropy Dynamics

Theorem 6.6.1. *The strength of entropy reduction is proportional to*

$$\frac{\epsilon(\epsilon-1)\left(\mathrm{e}^{\epsilon(\epsilon-1)(\gamma_{\neg T}-\gamma_T)}-\mathrm{e}^{\frac{\epsilon(\epsilon-1)\gamma_T(n-1)+\epsilon\gamma_{\neg T}(\epsilon(n-3)+n-1)}{(n-1)^2}}\right)}{(\epsilon-1)\,\mathrm{e}^{\epsilon(\epsilon-1)(\gamma_{\neg T}-\gamma_T)}-\epsilon\,\mathrm{e}^{\frac{\epsilon(\epsilon-1)\gamma_T(n-1)+\epsilon\gamma_{\neg T}(\epsilon(n-3)+n-1)}{(n-1)^2}}}. \tag{6.51}$$

Thus, values less than zero imply that entropy is increased, values greater than zero that it is decreased, and values equal to zero imply that there is no change.

Proof. Let us analyze the case where all non-target logits have the same value, $\frac{\epsilon}{n-1}$, and the target logit has the value $1-\epsilon$. That is, all non-target classes have equal probabilities.

A model's scaled logit for an input $\boldsymbol{x}_i$ can be represented as

$$h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) = \sigma_k(f(\boldsymbol{x}_i, \boldsymbol{\theta})) = \frac{\mathrm{e}^{f_k(\boldsymbol{x}_i, \boldsymbol{\theta})}}{\sum_{j=1}^{n} \mathrm{e}^{f_j(\boldsymbol{x}_i, \boldsymbol{\theta})}}, \tag{6.52}$$

where $f_k(\boldsymbol{x}_i, \boldsymbol{\theta})$ is a raw output logit from the model.

The (k, j)th entry of the Jacobian matrix for $h(\boldsymbol{x}_i, \boldsymbol{\theta})$ can be easily derived through application of the chain rule:

$$\boldsymbol{J}_{kj} h(\boldsymbol{x}_i, \boldsymbol{\theta}) = \frac{\partial h_k(\boldsymbol{x}_i, \boldsymbol{\theta})}{\partial f_j(\boldsymbol{x}_i, \boldsymbol{\theta})} = \begin{cases} h_j(\boldsymbol{x}_i, \boldsymbol{\theta}) \left(1 - h_k(\boldsymbol{x}_i, \boldsymbol{\theta})\right) f_k(\boldsymbol{x}_i, \boldsymbol{\theta}) & k = j \\ -h_j(\boldsymbol{x}_i, \boldsymbol{\theta}) \, h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) \, f_k(\boldsymbol{x}_i, \boldsymbol{\theta}) & k \neq j \; . \end{cases} \tag{6.53}$$

Consider an SGD learning rule of the form:

$$\theta_j \leftarrow \theta_j + \eta \frac{1}{n} \sum_{k=1}^{n} \left[\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}) D_{\boldsymbol{j}} \left(h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) \right) \right]. \tag{6.54}$$

Let us freeze a network at any specific point during the training process for any specific sample. Now, treat all $f_j(\boldsymbol{x}_i, \boldsymbol{\theta}), j \in [1, n]$ as free parameters with unit derivatives, rather than as functions; that is, $\theta_j = f_j(\boldsymbol{x}_i, \boldsymbol{\theta})$. Updates are now

$$\Delta f_j \propto \sum_{k=1}^{n} \gamma_j \begin{cases} h_j(\boldsymbol{x}_i, \boldsymbol{\theta}) \left(1 - h_k(\boldsymbol{x}_i, \boldsymbol{\theta})\right) & k = j \\ -h_j(\boldsymbol{x}_i, \boldsymbol{\theta}) \, h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) & k \neq j \; . \end{cases} \tag{6.55}$$

For downstream analysis, we can consider, as substitutions for γ_j above, $\gamma_{\neg T}$ to be the value for non-target logits, and γ_T for the target logit.

This sum can be expanded and conceptually simplified by considering j indices and $\neg j$ indices. The $\neg j$ indices, of which there are $n - 1$, are either all non-target logits, or one is the target logit in the case where j is not the target logit. Let us consider both cases, while substituting the scaled logit

values defined above:

$$\Delta f_j \propto \begin{cases} \gamma_{\neg T}\ \boldsymbol{J}_{k=j} h(\boldsymbol{x}_i, \boldsymbol{\theta}) + (n-2)\gamma_{\neg T}\ \boldsymbol{J}_{k\neq j} h(\boldsymbol{x}_i, \boldsymbol{\theta}) & \\ \qquad\qquad +\gamma_T\ \boldsymbol{J}_{k\neq j} h(\boldsymbol{x}_i, \boldsymbol{\theta}) & \text{non-target } j \\ \gamma_T\ \boldsymbol{J}_{k=j} h(\boldsymbol{x}_i, \boldsymbol{\theta}) + (n-1)\gamma_{\neg T}\ \boldsymbol{J}_{k\neq j} h(\boldsymbol{x}_i, \boldsymbol{\theta}) & \text{target } j\ . \end{cases} \tag{6.56}$$

$$\Delta f_j \propto \begin{cases} \gamma_{\neg T} h_{\neg T}(\boldsymbol{x}_i, \boldsymbol{\theta})\ (1 - h_{\neg T}(\boldsymbol{x}_i, \boldsymbol{\theta})) & \\ +(n-2)\gamma_{\neg T}\ (-h_{\neg T}(\boldsymbol{x}_i, \boldsymbol{\theta})\ h_{\neg T}(\boldsymbol{x}_i, \boldsymbol{\theta})) & \\ \qquad +\gamma_T\ (-h_{\neg T}(\boldsymbol{x}_i, \boldsymbol{\theta})\ h_T(\boldsymbol{x}_i, \boldsymbol{\theta})) & \text{non-target } j \\ \gamma_T h_T(\boldsymbol{x}_i, \boldsymbol{\theta})\ (1 - h_T(\boldsymbol{x}_i, \boldsymbol{\theta})) & \\ +(n-1)\gamma_{\neg T}\ (-h_{\neg T}(\boldsymbol{x}_i, \boldsymbol{\theta})\ h_T(\boldsymbol{x}_i, \boldsymbol{\theta})) & \text{target } j \end{cases} \tag{6.57}$$

$$\text{where} \quad h_T(\boldsymbol{x}_i, \boldsymbol{\theta}) = 1 - \epsilon, \quad h_{\neg T}(\boldsymbol{x}_i, \boldsymbol{\theta}) = \frac{\epsilon}{n-1} \tag{6.58}$$

$$\Delta f_j \propto \begin{cases} \gamma_{\neg T} \dfrac{\epsilon}{n-1}\left(1 - \dfrac{\epsilon}{n-1}\right) & \\ +\gamma_{\neg T}(n-2)\dfrac{\epsilon^2}{n^2 - 2n + 1} + \gamma_T(\epsilon - 1)\dfrac{\epsilon}{n-1} & \text{non-target } j \\ \gamma_T \epsilon - \gamma_T \epsilon^2 + \gamma_{\neg T}(n-1)(\epsilon - 1)\dfrac{\epsilon}{n-1} & \text{target } j \end{cases} \tag{6.59}$$

At this point, closed-form solutions for the changes to softmax inputs have been derived. To characterize entropy, solutions must be derived for the changes to softmax outputs given such changes to the inputs. That is:

$$\Delta\sigma_j(f(\boldsymbol{x}_i, \boldsymbol{\theta})) = \frac{\mathrm{e}^{f_j(\boldsymbol{x}_i, \boldsymbol{\theta}) + \Delta f_j}}{\sum_{k=1}^{n} \mathrm{e}^{f_k(\boldsymbol{x}_i, \boldsymbol{\theta}) + \Delta f_k}}. \tag{6.60}$$

Due to the two cases in Δf_j, $\Delta\sigma_j(f(\boldsymbol{x}_i, \boldsymbol{\theta}))$ is thus also split into two cases for target and non-target logits:

$$\Delta\sigma_j(f(\boldsymbol{x}_i, \boldsymbol{\theta})) = \begin{cases} \dfrac{\mathrm{e}^{f_{\neg T}(\boldsymbol{x}_i, \boldsymbol{\theta}) + \Delta f_{\neg T}}}{(n-1)\mathrm{e}^{f_{\neg T}(\boldsymbol{x}_i, \boldsymbol{\theta}) + \Delta f_{\neg T}} + \mathrm{e}^{f_T(\boldsymbol{x}_i, \boldsymbol{\theta}) + \Delta f_T}} & \text{non-target } j \\ \dfrac{\mathrm{e}^{f_T(\boldsymbol{x}_i, \boldsymbol{\theta}) + \Delta f_T}}{(n-1)\mathrm{e}^{f_{\neg T}(\boldsymbol{x}_i, \boldsymbol{\theta}) + \Delta f_{\neg T}} + \mathrm{e}^{f_T(\boldsymbol{x}_i, \boldsymbol{\theta}) + \Delta f_T}} & \text{target } j \end{cases} \tag{6.61}$$

Now, the observation can be made that scaled logits have a lower entropy distribution when $\Delta\sigma_T(f(\boldsymbol{x}_i, \boldsymbol{\theta})) > 0$ and $\Delta\sigma_{\neg T}(f(\boldsymbol{x}_i, \boldsymbol{\theta})) < 0$. Essentially, the target and non-target scaled logits are being repelled from each other. Either of these inequalities can be ignored, if one is satisfied then both are satisfied, in part because $|\boldsymbol{\sigma}(f(\boldsymbol{x}_i, \boldsymbol{\theta}))|_1 = 1$. The target-case constraint (i.e., the target scaled logit must grow) can be represented as

$$\frac{\mathrm{e}^{f_T(\boldsymbol{x}_i,\boldsymbol{\theta})+\Delta f_T}}{(n-1)\mathrm{e}^{f_{\neg T}(\boldsymbol{x}_i,\boldsymbol{\theta})+\Delta f_{\neg T}} + \mathrm{e}^{f_T(\boldsymbol{x}_i,\boldsymbol{\theta})+\Delta f_T}} > 1 - \epsilon. \tag{6.62}$$

Consider the target logit case prior to changes:

$$\frac{\mathrm{e}^{f_T(\boldsymbol{x}_i,\boldsymbol{\theta})}}{(n-1)\mathrm{e}^{f_{\neg T}(\boldsymbol{x}_i,\boldsymbol{\theta})} + \mathrm{e}^{f_T(\boldsymbol{x}_i,\boldsymbol{\theta})}} = 1 - \epsilon. \tag{6.63}$$

Let us solve for $\mathrm{e}^{f_T(\boldsymbol{x}_i,\boldsymbol{\theta})}$:

$$\begin{aligned} \mathrm{e}^{f_T(\boldsymbol{x}_i,\boldsymbol{\theta})} &= (n-1)\mathrm{e}^{f_{\neg T}(\boldsymbol{x}_i,\boldsymbol{\theta})} + \mathrm{e}^{f_T(\boldsymbol{x}_i,\boldsymbol{\theta})} - \epsilon(n-1)\mathrm{e}^{f_{\neg T}(\boldsymbol{x}_i,\boldsymbol{\theta})} - \epsilon\mathrm{e}^{f_T(\boldsymbol{x}_i,\boldsymbol{\theta})} && (6.64) \\ &= \left(\frac{n-1}{\epsilon} - n + 1\right)\mathrm{e}^{f_{\neg T}(\boldsymbol{x}_i,\boldsymbol{\theta})}. && (6.65) \end{aligned}$$

Substituting this definition into Equation 6.62:

$$\frac{\mathrm{e}^{\Delta f_T}\left(\frac{n-1}{\epsilon} - n + 1\right)\mathrm{e}^{f_{\neg T}(\boldsymbol{x}_i,\boldsymbol{\theta})}}{(n-1)\mathrm{e}^{f_{\neg T}(\boldsymbol{x}_i,\boldsymbol{\theta})+\Delta f_{\neg T}} + \mathrm{e}^{\Delta f_T}\left(\frac{n-1}{\epsilon} - n + 1\right)\mathrm{e}^{f_{\neg T}(\boldsymbol{x}_i,\boldsymbol{\theta})}} > 1 - \epsilon. \tag{6.66}$$

Coalescing exponents,

$$\frac{\mathrm{e}^{\Delta f_T + f_{\neg T}(\boldsymbol{x}_i,\boldsymbol{\theta})}\left(\frac{n-1}{\epsilon} - n + 1\right)}{(n-1)\mathrm{e}^{f_{\neg T}(\boldsymbol{x}_i,\boldsymbol{\theta})+\Delta f_{\neg T}} + \mathrm{e}^{\Delta f_T + f_{\neg T}(\boldsymbol{x}_i,\boldsymbol{\theta})}\left(\frac{n-1}{\epsilon} - n + 1\right)} + \epsilon - 1 > 0 \tag{6.67}$$

Substituting in definitions for Δf_T and $\Delta f_{\neg T}$ and greatly simplifying the left-hand side expression in a computer algebra system results in the removal of instances of $f_{\neg T}$:

$$\frac{\epsilon(\epsilon-1)\left(\mathrm{e}^{\epsilon(\epsilon-1)(\gamma_{\neg T}-\gamma_T)}-\mathrm{e}^{\frac{\epsilon(\epsilon-1)\gamma_T(n-1)+\epsilon\gamma_{\neg T}(\epsilon(n-3)+n-1)}{(n-1)^2}}\right)}{(\epsilon-1)\mathrm{e}^{\epsilon(\epsilon-1)(\gamma_{\neg T}-\gamma_T)}-\epsilon\mathrm{e}^{\frac{\epsilon(\epsilon-1)\gamma_T(n-1)+\epsilon\gamma_{\neg T}(\epsilon(n-3)+n-1)}{(n-1)^2}}} > 0 \tag{6.68}$$

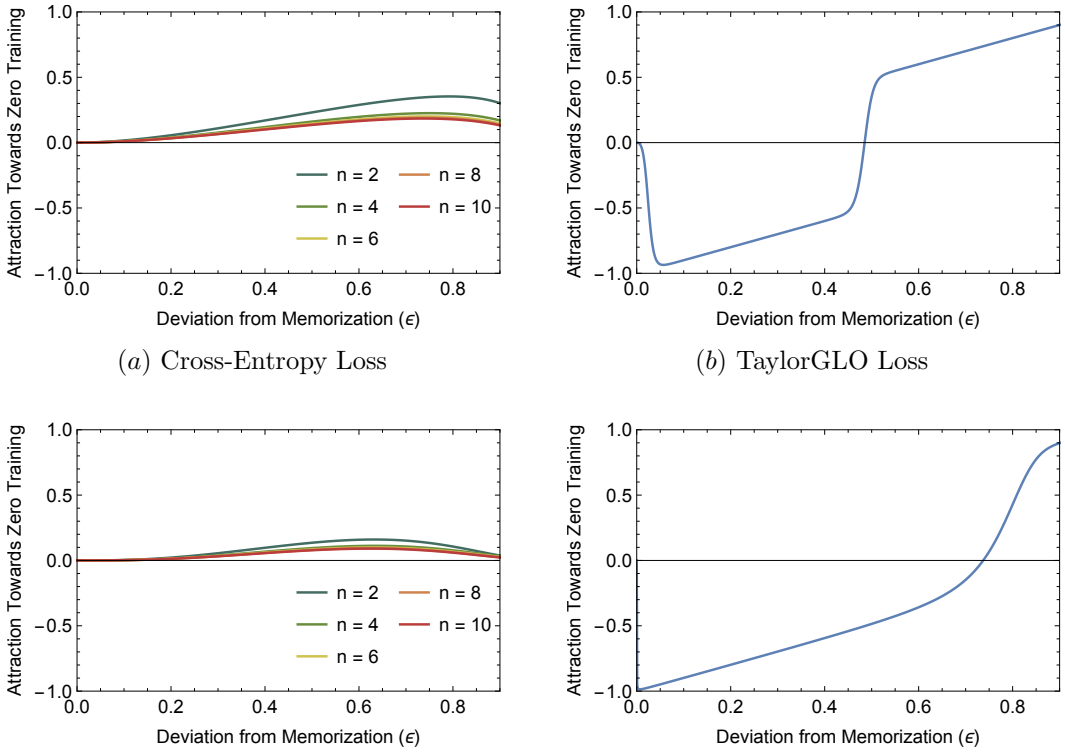
□

6.6.2 Zero Training Error Attractors

The strength of the entropy reduction in Theorem 6.6.1 can also be thought of as a measure of the strength of the attraction towards zero training error regions of the parameter space. For any given n, a value for this zero training error attraction strength at different ϵ values can be plotted using the corresponding γ_T and $\gamma_{\neg T}$ values from a particular loss function. These characteristic curves for four specific loss functions are plotted in Figure 6.1.

This strength can thus be calculated for individual training samples during any part of the training process, leading to the insight that the process results from competing "push" and "pull" forces. This theoretical insight, combined with empirical data from actual training sessions, explains how different loss functions balance data fitting and regularization.

Figure 6.2 provides one such example on AllCNN-C [161] models trained



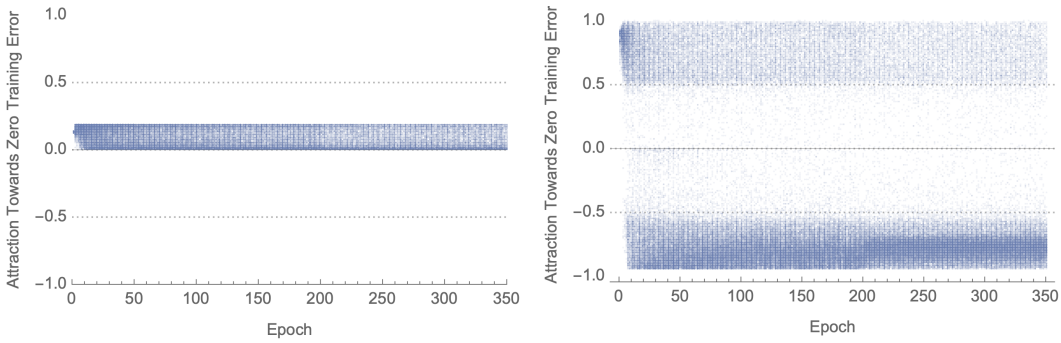
(*a*) Cross-Entropy Loss

(*b*) TaylorGLO Loss

(*c*) MSE Loss

(*d*) Baikal Loss

Figure 6.1: **Attraction towards zero training error curves with different loss functions.** Each loss function has a characteristic curve—plotted using Equation 6.51—that describes zero training error attraction dynamics for individual samples given their current deviation from perfect memorization, ϵ. Plots (a) and (b) only have the $n = 10$ case plotted, i.e. the 10-class classification case for which they were evolved. Cross-entropy (a) and MSE (c) loss functions have positive attraction for all values of ϵ. In contrast, the TaylorGLO loss function for CIFAR-10 on AllCNN-C (b) and the Baikal loss function (d) both have very strong attraction for weakly learned samples (on the right side), and repulsion for highly confidently learned samples (on the left side. This provides a graphical intuition for regularization with TaylorGLO and Baikal loss functions.



(a) Cross-Entropy Loss (b) TaylorGLO Loss

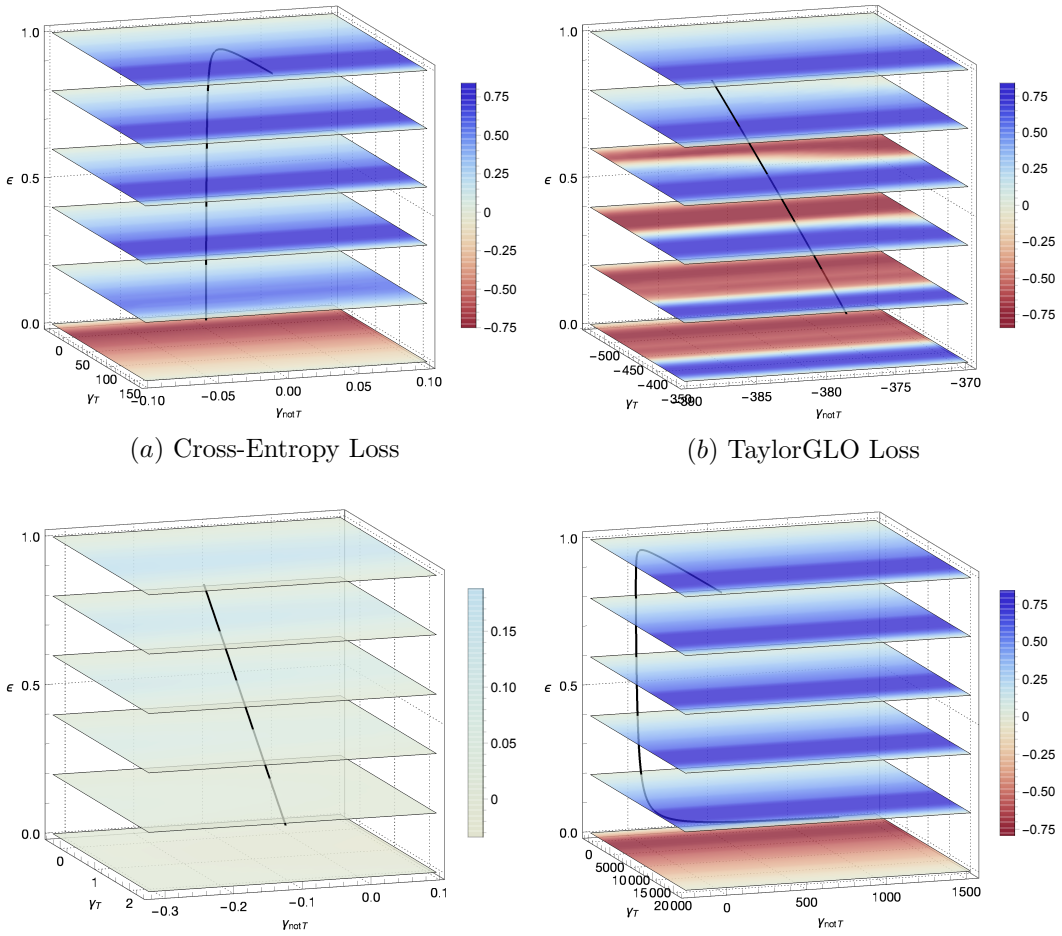
Figure 6.2: **Per-training-sample attraction towards zero training error with cross-entropy and TaylorGLO loss functions for CIFAR-10 AllCNN-C models.** Each point represents an individual training sample (500 are randomly sampled per epoch); its x-location indicates the training epoch, and y-location the strength with which the loss functions pulls the output towards the correct label, or pushes it away from it. With the cross-entropy loss (a), these values are always positive, indicating a constant pull towards the correct label for every single training sample. Interestingly, the TaylorGLO values (b) span both the positives and the negatives; at the beginning of training there is a strong pull towards the correct label (seen as the dark area on top left), which then changes to more prominent push away from it in later epochs (seen as the dark band on the bottom). This plot shows how TaylorGLO regularizes by preventing overconfidence and biasing solutions towards different parts of the weight space with higher performance.

on CIFAR-10 [94] with cross-entropy and custom TaylorGLO loss functions. Scaled target and non-target logit values were logged for every sample at every epoch and used to calculate respective γ_T and $\gamma_{\neg T}$ values. These values were then substituted into Equation 6.51 to get the strength of bias towards zero training error.

The cross-entropy loss exhibits a tendency towards zero training error for every single sample, as expected. The TaylorGLO loss, however, has a much different behavior. Initially, there is a much stronger pull towards zero training error for all samples—which leads to better generalization [104, 196]—after which a stratification occurs, where the majority of samples are repelled, and thus biased towards a different region of the weight space that happens to have better performance characteristics empirically. This, for the first time, explains theoretically why loss functions like Baikal serve as regularizers.

This measure of attraction can be seen as providing a value for every point in a phase space consisting of $\gamma_T, \gamma_{\neg T}, \epsilon$, and n. For a given loss function, γ_T and $\gamma_{\neg T}$ are implicit functions of the loss function at a certain ϵ deviation from memorization. Thus, each loss function encodes a characteristic, one-dimensional curve through this space, for a specific number of classes n. A visualization of these curves for four loss functions is provided in Figure 6.3. Note that the values at each point in each curve correspond to the plots in Figure 6.1.

Both GLO and TaylorGLO loss-function metalearning approaches optimize the loss function directly, and thus optimize the characteristic curve



(*a*) Cross-Entropy Loss

(*b*) TaylorGLO Loss

(*c*) MSE Loss

(*d*) Baikal Loss

Figure 6.3: **Characteristic curves in zero training error attraction phase space for different loss functions.** A general phase space for zero training error attraction can be constructed using a loss function's specific γ_T and $\gamma_{\neg T}$ values and a network's ϵ value for a given sample. Each loss function has a characteristic curve within this space. The values at each point in this space (i.e., colors) are calculated using Equation 6.51 and can be seen in Figure 6.1 for each loss function. While loss-function metalearning indirectly finds optimal characteristic curves, perhaps, in the future, these characteristic curves may be optimized directly.

through the attraction phase space indirectly. Perhaps, in future work, the process may be inverted, whereby these characteristic curves are optimized directly, and the loss function subsequently derived from it. This new method may serve as a way to more effectively discover different training behaviors, rather than a complex encoding of them, i.e. through a loss function.

Given that the values traced by each curve vary depending on the n for a given task, a given loss function can exhibit different behaviors for different numbers of classes. It should be possible to define a transformation on a loss function between any two values of n, such that it provides the same behavior at different ϵ values. This transformation can be thought of as a way to adapt metalearned loss functions to behave as intended in tasks with different numbers of classes as the task upon which the loss function was metalearned, assuming that the disparate tasks train similarly and have similarly distributed training samples.

6.7 Label Smoothing

TaylorGLO loss functions have been shown in the above sections to provide regularization through dynamic biases that are imparted throughout the training process. However, this behavior is not the only way that TaylorGLO can regularize. This section shows how the regularization imparted by label smoothing [167] can be implicitly represented by TaylorGLO.

Theorem 6.7.1. *For any third-order TaylorGLO loss function with $\boldsymbol{\lambda}$ parameters, there exists a TaylorGLO loss function defined by $\hat{\boldsymbol{\lambda}}$ that is equivalent*

in behavior to the original $\boldsymbol{\lambda}$ loss function when label smoothing is applied, for any label smoothing strength.

Proof. Consider a basic setup with standard label smoothing, controlled by a hyperparameter $\alpha \in (0, 1)$, such that the target value in any $\boldsymbol{y}_i$ is $1 - \alpha\frac{n-1}{n}$, rather than 1, and non-target values are $\frac{\alpha}{n}$, rather than 0. The learning rule changes in the general case as follows:

$$\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta}) = \begin{cases} c_1 + c_h h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) + c_{hh} h_k(\boldsymbol{x}_i, \boldsymbol{\theta})^2 \\ \quad + c_{hy} h_k(\boldsymbol{x}_i, \boldsymbol{\theta})\frac{\alpha}{n} + c_y\frac{\alpha}{n} + c_{yy}\frac{\alpha^2}{n^2} & y_{ik} = 0 \\ \\ c_1 + c_h h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) + c_{hh} h_k(\boldsymbol{x}_i, \boldsymbol{\theta})^2 \\ \quad + c_{hy} h_k(\boldsymbol{x}_i, \boldsymbol{\theta})\left(1 - \alpha\frac{n-1}{n}\right) \\ \quad + c_y\left(1 - \alpha\frac{n-1}{n}\right) + c_{yy}\left(1 - \alpha\frac{n-1}{n}\right)^2 & y_{ik} = 1\ . \end{cases} \tag{6.69}$$

Let $\hat{c}_1, \hat{c}_h, \hat{c}_{hh}, \hat{c}_{hy}, \hat{c}_y, \hat{c}_{yy}$ represent settings for $c_1, c_h, c_{hh}, c_{hy}, c_y, c_{yy}$ in the non-label-smoothed case that apply label smoothing implicitly within the TaylorGLO parameterization. Given the two cases in the label-smoothed and non-label-smoothed definitions of $\gamma_k(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta})$, there are two equations that must be satisfiable by settings of $\hat{c}$ constants for any c constants, with shared terms highlighted in blue and red:

$$\begin{aligned} &c_1 + c_h h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) + c_{hh} h_k(\boldsymbol{x}_i, \boldsymbol{\theta})^2 + c_{hy} h_k(\boldsymbol{x}_i, \boldsymbol{\theta})\frac{\alpha}{n} + c_y\frac{\alpha}{n} + c_{yy}\frac{\alpha^2}{n^2} \\ &\qquad = \hat{c}_1 + \hat{c}_h h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) + \hat{c}_{hh} h_k(\boldsymbol{x}_i, \boldsymbol{\theta})^2 \end{aligned} \tag{6.70}$$

$$
\begin{aligned}
&c_1 + c_h h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) + c_{hh} h_k(\boldsymbol{x}_i, \boldsymbol{\theta})^2 + c_{hy} h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) \left(1 - \alpha \frac{n-1}{n}\right) \\
&\qquad + c_y \left(1 - \alpha \frac{n-1}{n}\right) + c_{yy} \left(1 - \alpha \frac{n-1}{n}\right)^2 \\
&= \hat{c}_1 + \hat{c}_h h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) + \hat{c}_{hh} h_k(\boldsymbol{x}_i, \boldsymbol{\theta})^2 + \hat{c}_{hy} h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) + \hat{c}_y + \hat{c}_{yy}.
\end{aligned} \tag{6.71}
$$

Let us then factor the left-hand side of Equation 6.70 in terms of different powers of $h_k(\boldsymbol{x}_i, \boldsymbol{\theta})$:

$$
\underbrace{\left(c_1 + c_y \frac{\alpha}{n} + c_{yy} \frac{\alpha^2}{n^2}\right)}_{\hat{c}_1} + \underbrace{\left(c_h + c_{hy} \frac{\alpha}{n}\right)}_{\hat{c}_h} h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) + \underbrace{c_{hh}}_{\hat{c}_{hh}} h_k(\boldsymbol{x}_i, \boldsymbol{\theta})^2, \tag{6.72}
$$

resulting in definitions for $\hat{c}_1, \hat{c}_h, \hat{c}_{hh}$. Let us then add the following form of zero to the left-hand side of Equation 6.71:

$$
\left(c_{hy} h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) \frac{\alpha}{n} + c_y \frac{\alpha}{n} + c_{yy} \frac{\alpha^2}{n^2}\right) - \left(c_{hy} h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) \frac{\alpha}{n} + c_y \frac{\alpha}{n} + c_{yy} \frac{\alpha^2}{n^2}\right). \tag{6.73}
$$

This operation allows the definitions for $\hat{c}_1, \hat{c}_h, \hat{c}_{hh}$ from Equation 6.72 to be substituted into Equation 6.71:

$$
\begin{aligned}
&\hat{c}_1 + \hat{c}_h h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) + \hat{c}_{hh} h_k(\boldsymbol{x}_i, \boldsymbol{\theta})^2 - \left(c_{hy} h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) \frac{\alpha}{n} + c_y \frac{\alpha}{n} + c_{yy} \frac{\alpha^2}{n^2}\right) \\
&+ c_{hy} h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) \left(1 - \alpha \frac{n-1}{n}\right) + c_y \left(1 - \alpha \frac{n-1}{n}\right) + c_{yy} \left(1 - \alpha \frac{n-1}{n}\right)^2 \\
&= \hat{c}_1 + \hat{c}_h h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) + \hat{c}_{hh} h_k(\boldsymbol{x}_i, \boldsymbol{\theta})^2 + \hat{c}_{hy} h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) + \hat{c}_y + \hat{c}_{yy}.
\end{aligned} \tag{6.74}
$$

Simplifying into

$$
\begin{aligned}
&c_{hy} h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) \left(1 - \alpha \frac{n-1}{n}\right) + c_y \left(1 - \alpha \frac{n-1}{n}\right) + c_{yy} \left(1 - \alpha \frac{n-1}{n}\right)^2 \\
&\qquad - \left(c_{hy} h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) \frac{\alpha}{n} + c_y \frac{\alpha}{n} + c_{yy} \frac{\alpha^2}{n^2}\right) \\
&\qquad = \hat{c}_{hy} h_k(\boldsymbol{x}_i, \boldsymbol{\theta}) + \hat{c}_y + \hat{c}_{yy}.
\end{aligned} \tag{6.75}
$$

Finally, factor the left-hand side of Equation 6.75 in terms of, $h_k(\boldsymbol{x}_i, \boldsymbol{\theta})$, 1, and 1^2:

$$\begin{aligned} &\underbrace{\left(c_{hy}\left(1-\alpha\frac{n-1}{n}\right)-c_{hy}\frac{\alpha}{n}\right)}_{\hat{c}_{hy}} h_k(\boldsymbol{x}_i,\boldsymbol{\theta}) \\ &+\underbrace{\left(c_y\left(1-\alpha\frac{n-1}{n}\right)-c_y\frac{\alpha}{n}\right)}_{\hat{c}_y}+\underbrace{\left(c_{yy}\left(1-\alpha\frac{n-1}{n}\right)^2-c_{yy}\frac{\alpha^2}{n^2}\right)}_{\hat{c}_{yy}}. \end{aligned} \tag{6.76}$$

Thus, the in-parameterization constants with implicit label smoothing can be defined for any desired, label-smoothed constants as follows:

$$\hat{c}_1 = c_1 + c_y\frac{\alpha}{n} + c_{yy}\frac{\alpha^2}{n^2} \tag{6.77}$$

$$\hat{c}_h = c_h + c_{hy}\frac{\alpha}{n} \tag{6.78}$$

$$\hat{c}_{hh} = c_{hh} \tag{6.79}$$

$$\hat{c}_{hy} = c_{hy}\left(1-\alpha\frac{n-1}{n}\right) - c_{hy}\frac{\alpha}{n} \tag{6.80}$$

$$\hat{c}_y = c_y\left(1-\alpha\frac{n-1}{n}\right) - c_y\frac{\alpha}{n} \tag{6.81}$$

$$\hat{c}_{yy} = c_{yy}\left(1-\alpha\frac{n-1}{n}\right)^2 - c_{yy}\frac{\alpha^2}{n^2} \tag{6.82}$$

□

So for any $\boldsymbol{\lambda}$ and any $\alpha \in (0,1)$, there exists a $\hat{\boldsymbol{\lambda}}$ such that the behavior imposed by $\hat{\boldsymbol{\lambda}}$ without explicit label smoothing is identical to the behavior imposed by $\boldsymbol{\lambda}$ *with* explicit label smoothing. That is, any degree of label smoothing can be implicitly represented for any TaylorGLO loss function. Furthermore, the presented methodology can be inverted and used to decompose

a given TaylorGLO loss function into a non-label-smoothed variant and a corresponding α. Thus, TaylorGLO may discover and utilize label smoothing as part of discovering loss functions, increasing their ability to regularize further.

6.8 Discussion

Based on a decomposition of the SGD learning rule, this chapter developed a unified theoretical framework that illustrates how loss function regularization can be better understood. Analysis of their behavior at different stages of the training process helped characterize different loss functions. At the null epoch, all analyzed loss functions reduce training error, albeit with varying strengths. At zero training error—that is, when training labels are memorized perfectly—different loss functions have significantly different behaviors. When there is no longer anything left to learn, the optimization biases of each loss function become clear, demonstrating how they result in implicit regularization.

Extending to the general case—optimization at any epoch for any trainable parameter configuration—a constraint was developed on the proportional strength with which the network output distribution's entropy increase or decrease (Equation 6.51). This quantity can be thought of as a measure of repulsion or attraction towards regions of the parameter space with zero training error. Each loss function has a characteristic curve for this value at varying levels of memorization (Figure 6.1). When calculated for individual samples during real training runs with different loss functions (Figure 6.2), TaylorGLO

loss functions exhibit remarkably different behavior from the cross-entropy loss. TaylorGLO loss functions initially force training towards reduced error much more strongly than the cross-entropy loss, and after a number of epochs change to a different training regime where the majority of training samples are repel from zero training error regions. Since these characteristic curves all lie in the same phase space, future work could tackle metalearning these curves themselves rather than loss function, thus providing a more direct representation of loss function behavior.

There are many opportunities for further theoretical work in this area. Certain key questions remain unanswered, such as the relationship between model architecture and regularization, and will likely require theoreticians to develop a comprehensive theory of regularization and generalization in deep neural networks. However, the framework and analyses presented in this chapter provide a compelling first step, and have already shed light on the effectiveness of metalearned loss functions.

6.9 Conclusion

Regularization has long been a crucial aspect of training deep neural networks, with many different flavors. This chapter contributed an understanding of regularization resulting from loss-function metalearning. A theoretical framework for representing different loss functions was first developed in order to analyze their training dynamics in various contexts. The results demonstrate that TaylorGLO loss functions implement a guard against overfit-

ting, resulting in automatic regularization. The results thus extend the scope of metalearning, focusing it not just on finding optimal model configurations, but also on improving regularization, learning efficiency, and robustness directly. In Chapter 7, two practical opportunities emerge from these analyses: filtering based on an invariant derived at the null epoch will be shown to improve the search process, and the robustness against overfitting will be shown to make networks also more robust against adversarial attacks.

Chapter 7

TaylorGLO Extensions

TaylorGLO is formulated in a general manner and it can therefore be extended to address several different challenges and opportunities in machine learning. This chapter presents four such extensions: utilizing auxiliary loss functions, focussing the search process through an invariant, guard against adversarial attacks, and coevolving loss functions for GANs.

7.1 Auxiliary Classifier Loss Functions

As described in Section 2.5.3, auxiliary classifiers provide a unique form form of regularization [167] through architectural changes, while also alleviating the vanishing gradients problem and providing a way to more effectively learn lower-level features than in standard networks. This section characterizes the behavior of TaylorGLO on networks with auxiliary classifiers, focussing on a modified AllCNN-C network with two auxiliary classifiers. Three seperate loss functions are jointly evolved, which are able to take advantage of the three sets of network outputs.

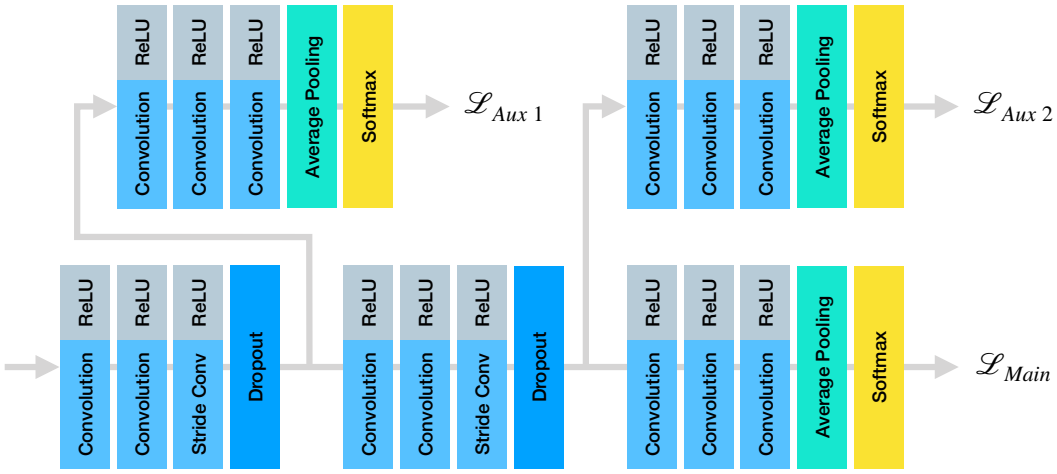


Figure 7.1: **AllCNN-C with Auxiliary Classifiers.** The AllCNN-C architecture can be augmented with auxiliary classifiers after each dropout layer to provide regularization [167] and allow gradients to flow deeper into the model more directly. TaylorGLO can improve the model's performance by optimizing three separate loss functions.

7.1.1 AllCNN-C with Auxiliary Classifiers

AllCNN-C provides two natural points where auxiliary classifiers may be grafted onto the architecture: it is possible to do so after each of the two dropout layers. Each auxiliary classifier is nearly identical to the final five layers of AllCNN-C, albeit with half as many filters. Specifically, both auxiliary classifier have the following layers: (1) 1×1 convolution with 96 filters and ReLU activations, (2) 1×1 convolution with 10 filters and ReLU activations, (3) average pooling, and (4) a softmax layer. The modified architecture is illustrated in Figure 7.1.

Auxiliary losses are scaled by 0.3 and summed with the main loss, resulting in the final training loss. The 0.3 scaling factor is appropriate

for the Inception-v3 architecture [167], and was confirmed to be appropriate for AllCNN-C in a hyperparameter sweep with varying scaling values in $\langle 0.15, 0.3, 1, 2 \rangle$.

AllCNN-C is the only architecture that was evaluated with auxiliary classifiers since it is simple and does not have skip connections, so that the effects of auxiliary classifiers can be seen more clearly. Auxiliary classifiers tend to be useful in deep networks without skip-connections, which provide a different way for gradients to propagate more deeply. In fact, in an experiment where auxiliary classifiers with the cross-entropy loss were added to a Pre ResNet-20, training failed in nearly half of all training attempts and resulted in slightly lower accuracy than the baseline when training did converge.

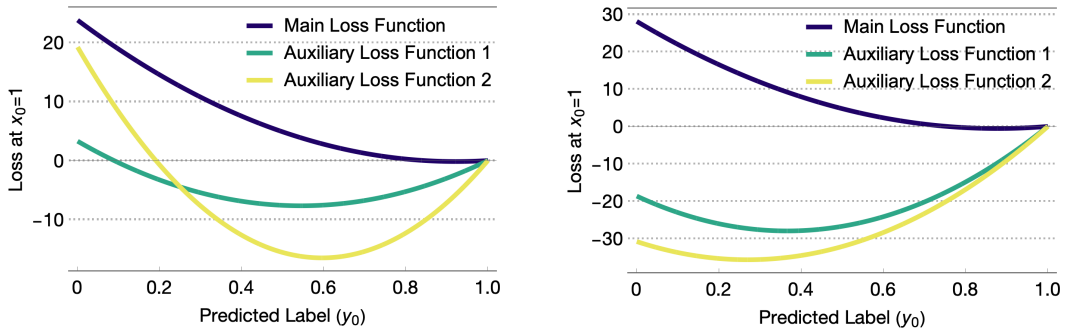
7.1.2 Experiments

Compared to the baseline, auxiliary classifiers provide an increase in accuracy greater than one percentage point (Table 7.1). Notably, this performance increase is not simply the result of a substantial increase in the number of parameters; the AllCNN-C variant with auxiliary classifiers has only 1.40 million parameters, compared to the conventional AllCNN-C's 1.37 million. The addition of Cutout has a similar effect on the modified network as on the baseline. There is a slight drop in performance, however the drop has a slightly smaller magnitude in the auxiliary classifier case.

Integration with TaylorGLO is fairly straightforward: the three loss functions' parameter sets are concatenated and jointly optimized as if they

Table 7.1: **Test-set performance of models with and without auxiliary classifiers.** Loss functions discovered by TaylorGLO are compared to the cross-entropy loss. The TaylorGLO results are based on the loss function with the highest validation accuracy during evolution. All averages are from ten separately trained models and p-values are from one-tailed Welch's t-Tests. Standard deviations are shown in parentheses. Auxiliary classifiers improve accuracy over the cross-entropy baseline. This improvement is further enhanced by TaylorGLO, where unique loss functions are evolved for each auxiliary classifer. An ablation study is also presented where the TaylorGLO loss function for the main output is also used for the auxiliary classifiers, weighted by 0.3. This study demonstrates the power of having separate loss functions. On AllCNN-C, the top result comes from the combination of Cutout regularization and TaylorGLO with auxiliary classifiers, suggesting that they each provide a different dimension of regularization.

Task and Model	Avg. TaylorGLO Acc.	Baseline Acc.	p-value
CIFAR-10			
AllCNN-C	**0.9271 (0.0013)**	0.8965 (0.0021)	0.42×10^{-17}
AllCNN-C + Cutout	**0.9329 (0.0022)**	0.8911 (0.0037)	1.60×10^{-14}
AllCNN-C + Aux (Ablation)	**0.9255 (0.0011)**	0.9112 (0.0022)	9.67×10^{-11}
AllCNN-C + Aux	**0.9307 (0.0021)**	0.9112 (0.0022)	1.03×10^{-13}
AllCNN-C + Aux + Cutout	**0.9369 (0.0010)**	0.9078 (0.0012)	5.36×10^{-21}



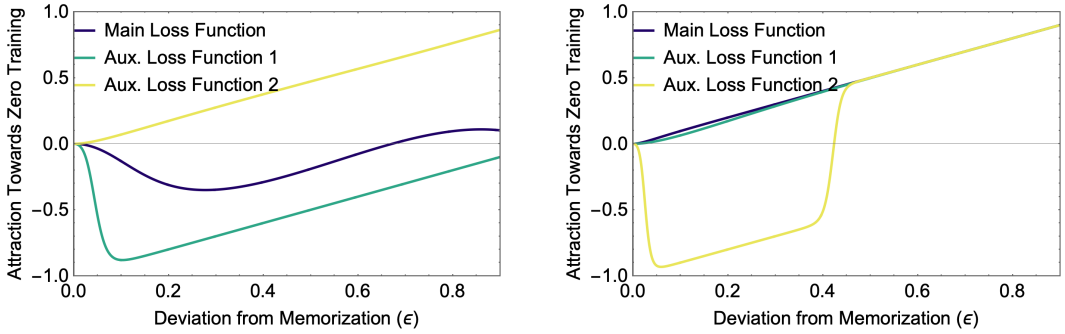
(*a*) Auxiliary Classifiers (*b*) Auxiliary Classifiers + Cutout

Figure 7.2: **Best TaylorGLO loss functions for AllCNN-C with auxiliary classifiers.** Loss functions are plotted for binary classification at $x_0 = 1$. Correct predictions lie on the right side of the graphs, and incorrect ones on the left. There is a clear difference in what is optimal for each of the auxiliary classifiers and the main loss function, both with and without Cutout. With Cutout, the optimal functions for auxiliary classifiers are noticably different, both in shape and range. Thus, what is optimal is influenced by both architecture and complementary regularization techniques.

were from a single loss function. To accommodate the larger search space, the population size is doubled from 20 to 40. The auxiliary loss functions' scaling factors are both set to one, with the intention that the proper scaling would be metalearned implicitly as part of TaylorGLO.

In every case in Table 7.1, TaylorGLO provides a statistically significant improvement in performance. TaylorGLO works harmoniously with other regularization techniques; the best result on AllCNN-C includes auxiliary classifiers, Cutout, and TaylorGLO.

The best set of discoverd loss functions is illustrated in Figure 7.2. The three loss functions are very different in terms of both shape and range, suggesting that they are specialized to each auxiliary classifier.



(*a*) Auxiliary Classifiers (*b*) Auxiliary Classifiers + Cutout

Figure 7.3: **Attraction towards zero training error curves for TaylorGLO loss functions for AllCNN-C with auxiliary classifiers.** Each curve—plotted using Equation 6.51—describes the zero training error attraction dynamics for individual samples given their current deviation from perfect memorization, ϵ. TaylorGLO is able to discover loss functions with fundamentally different training dynamics for each setting, demonstrating how TaylorGLO is able synergize with existing forms of regularization to reach higher levels of performance.

The set of three loss functions is also different when Cutout regularization is included; for example, they cover a larger range. The specific differences in behavior can be analyzed by looking at each loss function's characteristic zero training error attraction curve (Figure 7.3). Without Cutout, the main loss function only provides a force towards zero training error for sufficiently low levels of memorization, while the first auxiliary loss function always provides a repulsive effect away from zero training error, and the second auxiliary loss function always provides a force towards zero training error—like the cross-entropy loss does. Conversely, with Cutout, the main loss function and first auxiliary loss function always tends towards zero training error, and the second auxiliary loss function only provides an attracive force towards zero training

error for low levels of memorization—similar to the TaylorGLO loss function. The two sets of loss functions thus have fundamentally different training dynamics. This result shows how TaylorGLO adapts loss functions to specific settings, i.e. the inclusion or lack of Cutout regularization.

This application to auxiliary classifiers provides a practical example of how the TaylorGLO technique can be extended to evolve multiple mutually-optimized loss functions simultaneously. TaylorGLO can thus be used to amplify the beneficial effects of auxiliary classifiers.

7.2 Utilizing an Invariant to Guide Search

There are many different instances of $\boldsymbol{\lambda}$ for which models are untrainable. Given the training dynamics at the null epoch (characterized in Section 6.4), general constraints on $\boldsymbol{\lambda}$ were derived in Section 6.5).

The inverse of these constraints may be used as an invariant during loss function evolution. That is, they can be used to identify entire families of loss function parameters that are not usable, rule them out during search, and thereby make the search more effective.

7.2.1 Integration with TaylorGLO

Before each candidate $\boldsymbol{\lambda}$ is evaluated, it is checked for conformance to the invariant. If the invariant is violated, the algorithm can skip that candidate's validation training and simply assign a fitness of zero. However, due to the added complexity that the invariant imposes on the fitness landscape, a

Table 7.2: **Test-set accuracy of loss functions discovered by TaylorGLO with and without an invariant constraint on λ.** Models were trained on the loss function that had the highest validation accuracy during TaylorGLO evolution. All averages are from ten separately trained models and p-values are from one-tailed Welch's t-Tests. Standard deviations are shown in parentheses. The invariant allows focusing metalearning to viable areas of the search space, resulting in better loss functions.

Task and Model	TaylorGLO	**+ Invariant**	p-value
CIFAR-10			
AlexNet	0.7901 (0.0026)	**0.7933 (0.0026)**	0.0092
PreResNet-20	0.9169 (0.0014)	0.9164 (0.0019)	0.2827
AllCNN-C	0.9271 (0.0013)	**0.9290 (0.0014)**	0.0004
AllCNN-C + Cutout	0.9329 (0.0022)	**0.9350 (0.0014)**	0.0124
Wide ResNet 28-5	0.9548 (0.0015)	0.9540 (0.0016)	0.1323

larger population size is needed for evolution within TaylorGLO to be more stable. Practically, a doubling of the population size from 20 to 40 works well.

7.2.2 Experiments

Table 7.2 presents results from TaylorGLO runs with and without the invariant on the CIFAR-10 image classification benchmark dataset [94] with various architectures. Networks with Cutout [35] were also evaluated to show that TaylorGLO provides a different approach to regularization. Standard training hyperparameters from the references were used for each architecture. Notably, the invariant allows TaylorGLO to discover loss functions that have statistically significantly better performance in many cases, and never a detrimental effect. These results demonstrate that the theoretical invariant is useful

in practice, and should become a standard in TaylorGLO applications.

7.3 Optimizing Loss Functions Against Adversarial Attacks

As was described in Section 5.5.3, TaylorGLO loss functions discourage overconfidence, i.e. the resulting activations are less extreme and vary more smoothly with input. Such encodings are likely to be more robust against noise, damage, and other imperfections in the data and in the network execution. In the extreme case, they may also be more robust against adversarial attacks. This hypothesis will be tested experimentally in this section.

7.3.1 Adversarial Attacks

Adversarial attacks elicit incorrect predictions from a trained model by changing input samples in small ways that can even be imperceptible. They are generally classified as "white-box" or "black-box" attacks, depending on whether the attacker has access to the underlying model or not. Naturally, white-box attacks are more powerful at overwhelming a model.

One such white-box attack is the Fixed Gradient Sign Method (FGSM) [56]. It is very simple to conduct; following evaluation of a dataset, input gradients are taken from the network following a backward pass. The individual gradients have their sign calculated in an element-wise fashion, and are added to future network inputs with an ϵ scaling factor (note that this is a different ϵ than in Chapter 6) that determines the attack strength. Specifically, inputs

are modified as

$$\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i + \epsilon \operatorname{sign}\left(\nabla_{\boldsymbol{x}_i} \mathcal{L}(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\theta})\right). \tag{7.1}$$

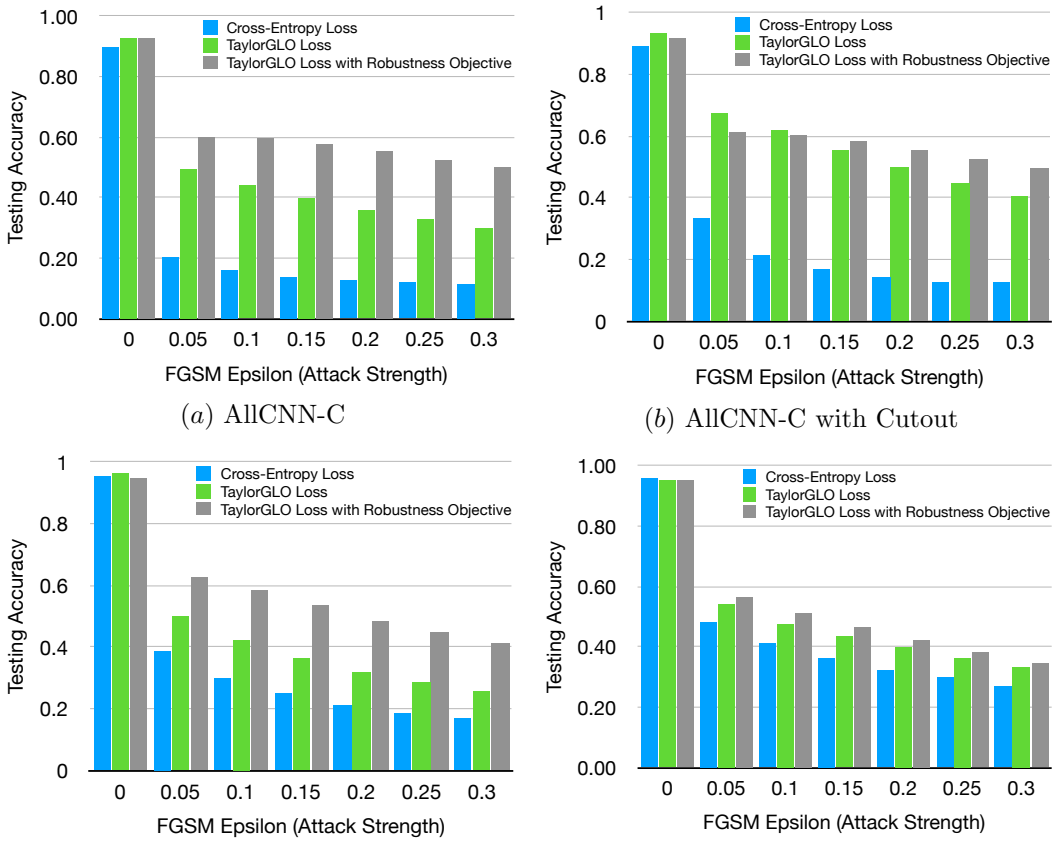
Experiments were designed to evaluate how well networks trained with TaylorGLO loss functions are able to handle FGSM attacks compared to those trained with the cross-entropy loss. Additionally, given that TaylorGLO can optimize loss functions against any objective, adversarially-hardened loss functions were evolved against an adversarial attack accuracy metric.

7.3.2 Experiments

Figure 7.4 shows how robust networks with different loss functions are to FGSM attacks of various strengths. In this experiment, AllCNN-C, AllCNN-C with Cutout, Wide ResNet 16-8, and Wide ResNet 28-5 networks were trained on CIFAR-10 with TaylorGLO and cross-entropy loss. Indeed, TaylorGLO outperforms the cross-entropy loss models significantly at all attack strengths.

Note that in this case loss functions were evolved simply to perform well, and adversarial robustness emerged as a side benefit. However, it is also possible to take adversarial attacks into account as an explicit objective in loss function evolution. Since TaylorGLO can uses non-differentiable metrics as objectives in its search process, the traditional validation accuracy objective can be replaced with validation accuracy at a particular FGSM attack strength.

Remarkably, loss functions found with this objective outperform both the previous TaylorGLO loss functions and the cross-entropy loss. These re-



(*a*) AllCNN-C

(*b*) AllCNN-C with Cutout

(*c*) Wide ResNet 16-8

(*d*) Wide ResNet 28-5

Figure 7.4: **Robustness of TaylorGLO loss functions against FGSM adversarial attacks on CIFAR-10.** For each architecture, the blue bars represent accuracy achieved through training with the cross-entropy loss, green bars with a TaylorGLO loss, and gray bars with a TaylorGLO loss specifically evolved in the adversarial attack environment. The leftmost points on each plot represent evaluations without adversarial attacks. TaylorGLO regularization makes the networks more robust against adversarial attacks, and this property can be further enhanced by making it an explicit goal in evolution.

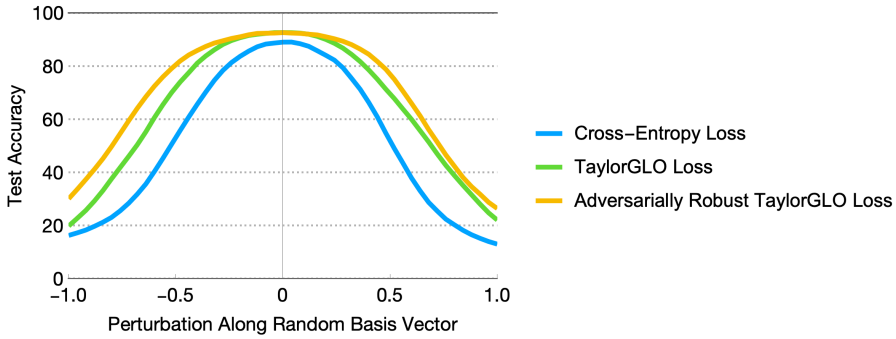


Figure 7.5: **Comparing accuracy basins of AllCNN-C with cross-entropy, TaylorGLO, and adversarially robust TaylorGLO loss functions on CIFAR-10.** Basins are only plotted along one perturbation direction for clarity, using the same technique as in Section 5.5.1. While the adversarially robust TaylorGLO loss function does not confer an increase in accuracy in the absence of adversarial attacks, it has a wider, flatter accuracy basin. This is indicative of increased robustness as a result of using a TaylorGLO loss function that has been selected for against an adversarial robustness objective.

sults demonstrate that the TaylorGLO regularization leads to robust encoding, and such robustness can be further improved by making it an explicit goal in loss-function optimization.

7.3.3 Comparing Accuracy Basins

As described in Section 5.5.1, models trained with TaylorGLO loss functions have flatter accuracy basins—a characteristic associated with robust networks. Indeed, as was shown above, they are robust even against adversarial attacks, presumably because of the flatter basins.

Since TaylorGLO loss functions that were discovered against an adversarial performance objective were even more robust, what do their basins

look like? On AllCNN-C, while the absolute accuracy on the testing set is not statistically significantly different when training with an adversarially robust versus a standard TaylorGLO loss function, the resultant accuracy basin is wider and flatter (Figure 7.5). This result suggests that it may be advantageous for TaylorGLO to evaluate against an adversarial performance metric, even in the absence of adversarial attacks in the target application.

7.4 Loss Functions for GANs

As GANs have grown in popularity, the difficulties involved in training them have become increasingly evident. The loss functions used to train a GAN's generator and discriminator networks greatly impact performance. Thus, optimizing these loss functions jointly can result in better GANs. This section presents an extension of TaylorGLO to evolve loss functions for GANs. Images generated in this way improve both visually and quantitatively, as the experiments in this section show.

7.4.1 TaylorGLO for GANs

In TaylorGLO for GANs, there are three functions that need to be jointly optimized (using the notation described in Table 2.1):

1. The component of the discriminator's loss that is a function of $D(\boldsymbol{x})$, the discriminator's output for a real sample from the dataset,

2. The synthetic / fake component of the discriminator's loss that is a

function of $D(G(\boldsymbol{z}))$, the discriminator's output from the generator that samples $\boldsymbol{z}$ from the latent distribution), and

3. The generator's loss, a function of $D(G(\boldsymbol{z}))$.

The discriminator's full loss is simply the sum of components (1) and (2). Table 7.3 shows how existing GAN formulations can be broken down into this tripartite loss.

Table 7.3: **GLO interpretation of existing GAN formulations.** These three components are all that is needed to define the discriminator's and generator's loss functions (sans regularization terms). Thus, TaylorGLO can discover and optimize new GAN formulations by jointly evolving three separate functions.

Formulation	**Loss D (real)** $\mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_{\text{data}}}$	**Loss D (fake)** $\mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_z}$	**Loss G (fake)** $\mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_z}$
GAN (minimax) [55]	$-\log D(\boldsymbol{x})$	$-\log(1-D(G(\boldsymbol{z})))$	$\log(1-D(G(\boldsymbol{z})))$
GAN (non-saturating) [55]	$-\log D(\boldsymbol{x})$	$-\log(1-D(G(\boldsymbol{z})))$	$-\log D(G(\boldsymbol{z}))$
WGAN [10]	$-D(\boldsymbol{x})$	$D(G(\boldsymbol{z}))$	$-D(G(\boldsymbol{z}))$
LSGAN [112]	$\frac{1}{2}(D(\boldsymbol{x})-1)^2$	$\frac{1}{2}(D(G(\boldsymbol{z})))^2$	$\frac{1}{2}(D(G(\boldsymbol{z}))-1)^2$

These three functions can be jointly evolved in a similar manner to that for auxiliary classifiers (Section 7.1). However, unlike classifier loss functions, GAN loss functions have a single input, i.e. $D(\boldsymbol{x})$ or $D(G(\boldsymbol{z}))$. Thus, a set of three third-order TaylorGLO loss functions for GANs requires only 12 parameters to be optimized, making the technique quite efficient.

Fitness for each set of three functions requires a different interpretation than in regular TaylorGLO. Since GANs cannot be thought of as having an

accuracy, a different metric needs to be used. The choice of fitness metric depends on the type of problem and target application. In the uncommon case where the training data's sampling distribution is known, the clear choice is the divergence between such a distribution and the distribution of samples from the generator. This approach will be used in the experiments below.

7.4.2 Experimental Setup

Rather than extending Lossferatu to support GANs, TaylorGLO for GANs was integrated into the LEAF evolutionary AutoML framework [105]. TaylorGLO parameters were evolved by the LEAF genetic algorithm as if they were hyperparameters. The implementation of CoDeepNEAT [115] for neural architecture search in LEAF was not used.

TaylorGLO for GANs was evaluated on the CMP Facade [176] dataset with a pix2pix-HD model [185]. The dataset consists of only 606 perspective-corrected 256×256 pixel images of building facades. Each image has a corresponding annotation image that segments facades into twelve different components, such as windows and doors. The objective is for the model to take an arbitrary annotation image as an input, and generate a photorealistic facade as output. The dataset was split into a training set with 80% of the images, and validation and testing sets, each with a disjoint 10% of the images.

Since individual image quality metrics can be exploited by adversarially constructed, lesser-quality images [20], two metrics were used to evaluate loss function candidates: (1) structural similarity index measure (SSIM) [186] be-

tween generated and ground-truth images, and (2) perceptual distance, implemented as the L_1 distance between VGG-16 [157] ImageNet [145] embeddings for generated and ground-truth images. During evolution, a composite objective [156] of these two metrics was used to evaluate candidates. The metrics were normalized (i.e., SSIM was multiplied by 17 and perceptual distance by -1) to have a similar impact on evolution.

The target GAN model, pix2pix-HD, is a refinement of the seminal pix2pix model [85]. Both models generate images conditioned upon an input image. Thus, they are trained with paired images. The baseline was trained with the Wasserstein loss [10] and spectral normalization [118] to enforce the Lipschitz constraint on the discriminator. The pix2pix-HD model is also trained with additive perceptual distance and discriminator feature losses. Both additive losses are multiplied by ten in the baseline. Models were trained for 60 epochs.

When running TaylorGLO experiments, each of the twleve TaylorGLO parameters was evolved within $[-10, 10]$. The learning rate and weights for both additive losses were also evolved since the baseline values, which are optimal for the Wasserstein loss, may not necessarily be optimal for TaylorGLO loss functions.

7.4.3 Experiments

TaylorGLO found a set of loss functions that outperformed the original Wasserstein loss with spectral normalization. After 49 generations of evolu-

Figure 7.6: **Five random samples from the CMP Facade test dataset, comparing Wasserstein and TaylorGLO loss functions.** The loss functions are used to train pix2pix-HD models that take architectural element annotations (top row) and generate corresponding photorealistic images similar to the ground-truth (second row). Images from the model trained with TaylorGLO (bottom row) have a higher quality than the baseline (third row). TaylorGLO images have more realistic coloration and finer details than the baseline.

tion, it discovered the loss functions

$$\begin{aligned}\mathcal{L}_{D_{\text{real}}} = 5.6484\ (D(\boldsymbol{x}) - 8.3399) + 9.4935\ (D(\boldsymbol{x}) - 8.3399)^2 \\ +8.2695\ (D(\boldsymbol{x}) - 8.3399)^3\end{aligned} \tag{7.2}$$

$$\begin{aligned}\mathcal{L}_{D_{\text{fake}}} = 6.7549\ (D(G(\boldsymbol{z})) - 8.6177) + 2.4328\ (D(G(\boldsymbol{z})) - 8.6177)^2 \\ +8.0006\ (D(G(\boldsymbol{z})) - 8.6177)^3\end{aligned} \tag{7.3}$$

$$\begin{aligned}\mathcal{L}_{G_{\text{fake}}} = 0.0000\ (D(G(\boldsymbol{z})) - 5.2232) + 5.2849\ (D(G(\boldsymbol{z})) - 5.2232)^2 \\ +0.0000\ (D(G(\boldsymbol{z})) - 5.2232)^3.\end{aligned} \tag{7.4}$$

A learning rate of 0.0001, discriminator feature loss weight of 4.0877, and perceptual distance loss weight of 10.3155 evolved for this candidate.

Figure 7.6 compares images for five random test samples that were generated with both the Wasserstein baseline and metalearned TaylorGLO loss functions. Visually, the TaylorGLO samples have more realistic coloration and details than the baseline. Baseline images all have an orange tint, while TaylorGLO images more closely match ground-truth images' typical coloration. Note that color information is not included in the model's input, so per-sample color matching is not possible. Additionally, TaylorGLO images tend to have higher-quality fine-grained details. For example, facade textures are unnaturally smooth and clean in the baseline, almost appearing to be made of plastic.

Quantitatively, the TaylorGLO model also outperforms the Wasserstein baseline. Across ten Wasserstein baseline runs, the average test-set SSIM was 9.4359 and the average test-set perceptual distance was 2129.5069. The TaylorGLO model improved both metrics, with a SSIM of 11.6615 and perceptual distance of 2040.2561.

Notably, the training set is very small, with fewer than 500 image pairs, showing how loss-function metalearning's benefits on small classification datasets also extend to GANs. Thus, metalearned loss functions are an effective way to train better GAN models, extending the types of problems to which TaylorGLO can be applied.

7.5 Conclusion

The TaylorGLO approach to evolving loss functions provides a practical foundation that can be extended to take advantage of various opportunities and to meet various challenges in machine learning. On model architectures with auxiliary classifiers, TaylorGLO can evolve unique loss functions for each classifier, allowing regularization to be tuned for different parts of the network. Leveraging the theoretical work in Chapter 6, an invariant on TaylorGLO parameters can be used to guide the evolutionary search process towards better regions of the loss function parameter landscape. Networks can be hardened against adversarial attacks by using metalearned loss functions; a property that can be amplified by using an adversarial fitness metric in the TaylorGLO search process. Finally, TaylorGLO is extended to evolve new formulations for conditional generative adversarial networks that provide higher quality image reconstructions. Altogether, these techniques provide a taste of what is possible with TaylorGLO, and loss-function metalearning more broadly.

Chapter 8

Discussion and Future Work

This dissertation introduced loss-function metalearning—a new type of metalearning that aims to optimize a given model's core training objective to improve performance along a variety of dimensions. This chapter provides a retrospective and discussion of the techniques, results, and analyses that have been presented. A prescription for AI practitioners is given, detailing how two new techniques, GLO and TaylorGLO, can be used in practice. Finally, the broader impacts of this research are discussed.

8.1 Loss-Function Metalearning

The research in this dissertation started from a basic hypothesis: Loss functions can be optimized automatically to find customized functions that perform better than those designed by hand. Verifying this hypothesis required the development of a technique to search a large space of loss functions effectively and methodically. Evolutionary computation was leveraged and ultimately this hypothesis was found valid with *Genetic Loss-function Optimization* (GLO), a general technique for loss-function metalearning.

GLO performed well on image classification models. Models trained

with a newly discovered loss function, Baikal, reached higher accuracies than those trained with cross-entropy loss more quickly. Additionally, Baikal functioned more effectively on small datasets. When training with increasingly small subsets of the training dataset, accuracy degraded less with Baikal than with cross-entropy loss.

Baikal was found to have a unique shape with an unintuitive property: loss values increase with increasingly confident predictions. This property may seem detrimental to proper training, but a distributional analysis of the network predictions showed that overly confident predictions are penalized, providing a form of regularization.

GLO discovered Baikal, and other performant loss functions, through a two-phase approach to loss-function metalearning: (1) new loss functions are evolved by having their structure represented as trees, allowing optimization through Genetic Programming (GP); and (2) loss function coefficients are optimized using a Covariance-Matrix Adaptation Evolutionary Strategy (CMA-ES). A variety of loss functions can be represented in this manner, since the GP search space can be augmented with new operators and loss function input nodes.

While the approach was successful, the separation of the two phases makes it challenging to find a mutually optimal structure and coefficients. Furthermore, small changes in the tree-based search space do not always result in small changes in the phenotype, and can easily make a function invalid, making the search process ineffective. To make loss-function metalearning

more practical, an alternative technique was developed: *Multivariate Taylor expansion-based genetic loss-function optimization* (TaylorGLO). TaylorGLO was designed with the intention of being a single-phase approach that finds performant loss functions with fewer candidate evaluations than GLO. By designing a more focused search space that sacrifices a degree of flexibility, evolution can optimize loss-functions more effectively in practice.

TaylorGLO evolves fixed-length vectors of values that define a loss function. With a novel parameterization for loss functions based on multivariate Taylor polynomials, the key pieces of information that affect a loss function's behavior are represented in these vectors compactly. Such vectors are then optimized for a specific task using CMA-ES. This optimization process is effective since the parameterization provides a smooth, well-behaved fitness landscape. This smoothness is in contrast to the fitness landscape in the structural evolution phase of GLO, where seemingly small changes to a function's structure can have disproportionate effects on fitness.

Because TaylorGLO is computationally more efficient, it can be applied to much larger models, with up to millions of trainable parameters. Several modern deep architectures were evaluated on three image classification benchmark datasets. Loss functions discovered by TaylorGLO for these tasks tend to outperform the cross-entropy loss statistically significantly. TaylorGLO finds loss functions that are customized to each setting, beating the "one size fits all" approach taken with the cross-entropy loss. They also outperform the Baikal loss, discovered by the original GLO technique, and do it with signifi-

cantly fewer function evaluations. As with Baikal, the reason for the improved performance is that evolved functions discourage overfitting to the class labels, thereby resulting in automatic regularization.

The power of TaylorGLO was demonstrated in a number of specialized settings. First, it was applied to models with auxiliary classifiers, where three separate loss functions were evolved. TaylorGLO took advantage of the differences between the different classifiers to evolve loss functions with unique behaviors. Overall, models with auxiliary classifiers trained with TaylorGLO had higher performance.

In a second special setting, TaylorGLO was adapted to evolve tripartite loss functions for conditional generative adversarial networks (GANs). On a facade generation dataset, TaylorGLO loss functions trained GANs that generated images that were quantitatively better than those produced from a GAN trained with a Wasserstein loss. This unique application showcases the power and flexibility of loss-function metalearning.

While TaylorGLO outperforms the original GLO technique in many practical applications, GLO is still useful on its own. In cases where a loss function may have several different inputs, GLO scales better than TaylorGLO, which would require a very large increase in the number of parameters (as described in Equation 5.4). Loss functions that include many different inputs, such as batch statistics, can represent richer behaviors during training by taking more data into account when optimizing a model. In such cases, GLO provides a possible approach that can find such loss functions by expanding its

search space. Thus, both GLO and TaylorGLO contribute unique capabilities to the field.

Notably, loss-function metalearning improves performance on networks that have already been tuned extensively by hand to perform well with cross-entropy loss. Thus, these designs may be overfit against the cross-entropy loss, potentially eclipsing some of the benefits that TaylorGLO loss functions can offer. Finding mutually beneficial architectures and loss functions is thus a promising direction for future work. One way this mutual optimization could be accomplished is by integrating TaylorGLO loss-function metalearning into a technique such as CoDeepNEAT [115].

Another important direction of future work is to apply loss-function metalearning to different types of problems. For example, the techniques can be easily applied to simple regression tasks, where different loss functions, such as the Huber loss [80], have been beneficial. On the other extreme, transformer models, such as BERT [34], which have been used to build state-of-the language models, are often fine-tuned to specific tasks with the cross-entropy loss. GLO and TaylorGLO can offer a way to evolve loss functions that are customized to each individual transformer architecture and target task, thus increasing their performance.

Overall, this dissertation presented results that validate loss-function metalearning as a technique to improve model performance. However, developing an understanding of the principles behind loss-function metalearning—that is, altered training dynamic and regularization—is also important.

8.2 Loss-Function Regularization and Effects on Models

In addition to showing that loss-funcion metalearning works from an empirical perspective, this dissertation provided a conceptual and theoretical understanding of the regularizing effects that loss functions impart on the training process.

To characterize networks trained with TaylorGLO loss functions, the minima they reach were analyzed. Using a prior loss landscape visualization technique that plots network performance through a random, filter-normalized slice of trainable parameter space [102], TaylorGLO loss functions were observed to result in flatter, lower minima. This finding indicates increased robustness and better generalization [90] compared to the cross-entropy loss.

While these minima had different performance characteristics, they were also in different parts of the trainable parameter space. This was demonstrated by analyzing the distribution of weights in networks trained with a TaylorGLO loss function and with the cross-entropy loss. Models trained with TaylorGLO had normally distributed weights, while the cross-entropy loss resulted in a Laplace distribution. Thus, the two loss functions reach entirely different regions of the parameter space. TaylorGLO models also had a significantly higher L_2 parameter norm than cross-entropy loss models. This finding helps further disprove [48] the common belief that networks with smaller parameter norms perform better [125, 191]. Additionally, these observed distributions and parameter norms were not greatly affected by the addition of Cutout regularization, indicating a different mechanism of regularization.

To understand how TaylorGLO loss functions reach different minima than the cross-entropy loss, networks' scaled logits were plotted in histograms throughout the training process. These visualizations showed qualitatively different training behavior between TaylorGLO and cross-entropy loss functions. Furthermore, the TaylorGLO loss functions that vary between architectures have very distinct behaviors, showing how TaylorGLO can customize and optimize training for each setting.

Following these empirical findings, a theoretical framework was developed to understand conceptually the specific behaviors that different loss functions effect. A novel decomposition of the stochastic gradient descent learning rule allows any loss function to be decomposed into two expressions that define the optimization process's behavior for any given sample. Using this framework, loss functions were characterized in a zero training error regime. In this case, where there is nothing left to learn from training data, the implicit biases that loss functions place on the optimization process become the only visible behavior. Metalearned loss functions represent a wide variety of implicit biases, showing one aspect of metalearned loss function regularization.

Subsequently, loss functions were characterized at the opposite end of the training process, i.e. at the null epoch. In this scenario, all loss functions fit the training data, with TaylorGLO loss functions offering flexibility in how strongly the data is fit. However, TaylorGLO is also able to represent undesired behaviors at the null epoch. These degenerate behaviors were characterized, and an invariant on TaylorGLO loss function parameters was developed using

the theoretical framework by setting a constraint on optimization behavior at the null epoch. This invariant was integrated into TaylorGLO, where it was used to avoid unnecessary candidate evaluations and guide evolution towards more fruitful areas of the search space. In this process, TaylorGLO found better loss functions that further improve performance in most cases.

The null epoch and zero training error represent the two extrema of the training process: no data fitting and perfect data fitting. To analyze training between these extrema, the way that a model's output distribution's entropy changed was characterized. A constraint was developed to determine the proportional strength of entropy growth or reduction for any given sample. Practically, the way entropy changes indicates whether a model's trainable parameters are being attracted or repelled from regions with zero training error. For each loss function, this value can be calculated for varying degrees of sample memorization. These values form a characteristic curve that represents the core data-fitting dynamics for a given loss function.

It turned out that the cross-entropy and mean-squared-error loss functions are always attracted towards zero training error, while TaylorGLO and Baikal loss functions are repelled for sufficiently well-memorized samples. This property explains how these metalearned loss functions regularize by preventing overconfidence.

The strength of attraction towards zero training error was then calculated for individual training samples during a real training processes. The cross-enropy loss, as expected, biased the model towards zero training error

for every sample, with a strength that decreased as training progressed. Conversely, the analyzed TaylorGLO loss function had a very strong bias towards zero training error in early training. As training progressed, well-fit samples were repelled from zero training error, while misclassified and poorly-fit samples were still strongly biased towards zero training error.

In the future, these entropy behavior characteristic curves can be evolved directly. Such curves represent a loss function's attraction or repulsion towards zero training error for varying levels of sample memorization. Thus, these behaviors can thus be directly metalearned—rather than being implicitly encoded in a loss function—by optimizing the characteristic curves directly.

Overall, these analyses provide a unique way to observe different loss functions' behavior. Different loss functions are seen to work in slightly different ways, customizing regularization to each domain. As such, the interaction between loss function regularization and other regularization technique may vary between models. These interactions are explored next.

8.3 Interactions Between Different Regularization Methods

This dissertation evaluated loss-function metalearning in many different settings with different types of regularization techniques. The conclusion is that metalearned loss functions provide regularizing effects in a way that is different from other regularization techniques.

Different regularization techniques interact in different ways and with

metalearned loss functions and the target model architecture. These differences provide evidence that regularization is not necessarily a continuum. Thus, value judgements on quantities of regularization, such as the assertion that "a model has more or less regularization than another model," are oversimplifications. Individual regularization techniques must instead be thought of as different processes that need to be compared comprehensively.

Many types of regularization can be viewed as operations that are added to a base network, each with a unique set of performance characteristics. Differences between different combinations of operations can be compared using interaction graphs. The base architecture lies at the center of the graph. Following each edge away from the center adds a specific type of regularization. The color of each edge characterizes the change in performance that results from the addition: green is a statistically significant, positive improvement, red is a statistically significant, negative improvement, and yellow indicates a statistically insignificant difference. A particularly useful characteristic of these graphs is that there are multiple instances of nodes for a given set of operations, one for each possible ordering the addition of these operations. These multiple paths provides insights into the interactions between different regularization techniques.

Figure 8.1 shows such an interaction graph for the AllCNN-C architecture. To construct this graph, experiments were run with ten repetitions each to collect performance statistics. Several interesting observations can be made on the graph. For example, on the AllCNN-C architecture (Figure 8.1),

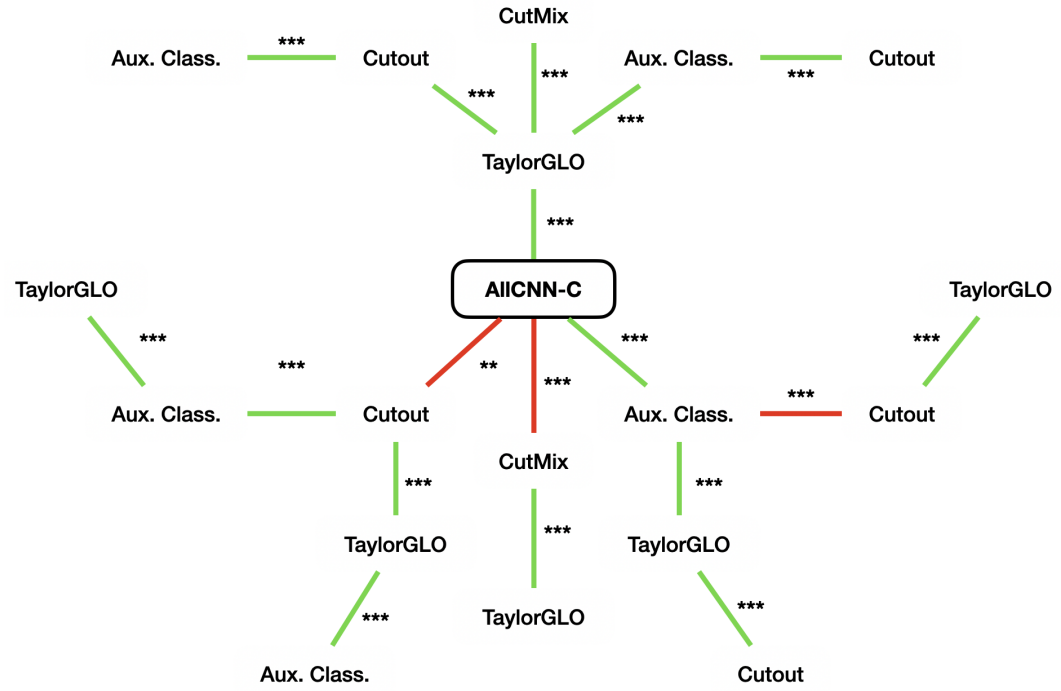


Figure 8.1: **Regularization interaction graph for AllCNN-C on CIFAR-10.** Each consecutive node away from the center node represents the addition of a single regularization technique. The edges leading to these nodes are colored green if the technique improves performance compared to the previous node, and red if they are detrimental, and yellow (in Figure 8.3) if there is no effect. On AllCNN-C, all techniques improve performance, with the notable exception of Cutout and CutMix, which only improve performance when coupled with TaylorGLO. This shows how regularization techniques are not necessarily additive operations; interactions between regularization techniques are more complex.

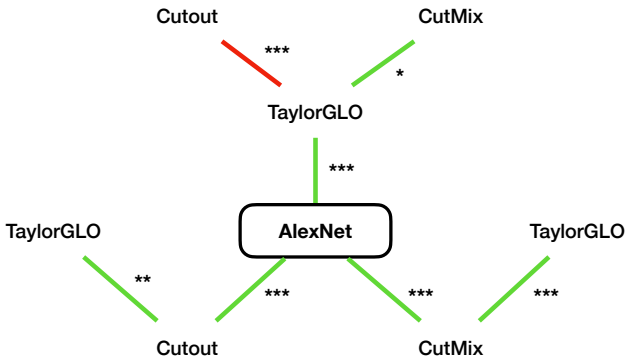
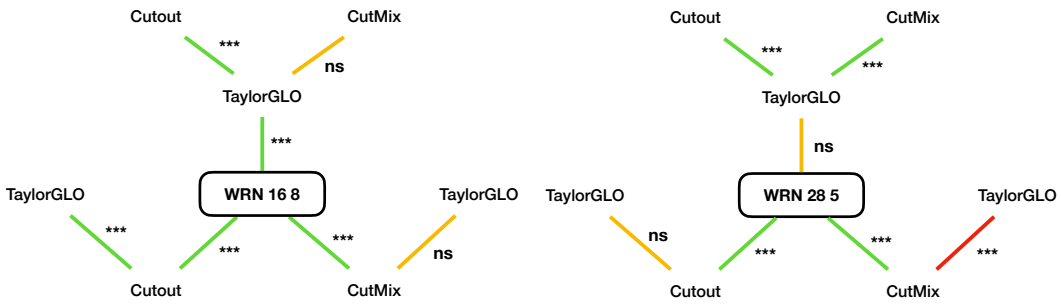


Figure 8.2: **Regularization interaction graph for AlexNet on CIFAR-10.** On AlexNet, all tested regularization techniques improves performance, except for Cutout alone. However, it also exhibits a constructive interaction with TaylorGLO, much like on AllCNN-C. Unlike on AllCNN-C, CutMix improves performance in the abscence of TaylorGLO. These differences show that even superficially similar architectures are affected by regularization differently.

the addition of Cutout always results in a performance drop unless it follows the addition of TaylorGLO. This property indicates a constructive relationship between TaylorGLO and Cutout. Such a relationship can be intuited from the snowflake plots by observing patterns in the paths leading to Cutout nodes. The same can be seen for CutMix. All other evaluated regularization techniques exhibit additive effects.

On the AlexNet architecture (Figure 8.2), Cutout alone is detrimental, but exhibits the same constructive interaction behavior with TaylorGLO, as on AllCNN-C.

On Wide ResNets (Figure 8.3), a modern family of architectures with skip connections, CutMix and TaylorGLO have a complex relationship. On



(a) Wide ResNet 16-8

(b) Wide ResNet 28-5

Figure 8.3: **Regularization interaction graphs for Wide ResNets on CIFAR-10.** On wide and shallow networks (a), all regularization techniques garner performance improvements, while combining CutMix and TaylorGLO does not significantly alter performance. Conversely, narrow and deep networks exhibit very different interactions, in that TaylorGLO does not improve performance significantly on its own. Ultimately, altering the depth and width of a single type of architecture affects the way in which regularization can happen. The similarities and differences in Figures 8.1 through 8.3 show how the interactions between different regularization techniques depend on the model architecture. They show that regularization techniques are not simply additive behaviors; their impacts on performance depend on which other regularization techniques are present.

Wide ResNet 16-8, TaylorGLO and CutMix, individually, improve the trained model's performance significantly. However, combining both does not result in any additional performance improvements. Additionally, unlike on AllCNN-C and AlexNet, Cutout always improves performance. However, on Wide ResNet 28-5, a variant that is more deep and narrow, the regularization interactions are quite different, in that TaylorGLO ceases to improve performance significantly.

A deeper understanding of these interferences, and why certain regularization techniques function on certain architectures but not on others, is a compelling direction for future work. While developing a comprehensive theoretical framework for regularization would be very challenging—due to the varied ways in which different regularization techniques function—perhaps understanding specific interferences yields insights that lead towards such a framework.

From a practical perspective, the interactions between different regularization techniques can be leveraged by future metalearning techniques to find optimal combinations for a given scenario. On the other hand, practitioners should not assume that regularization techniques will always be beneficial. This further demonstrates the importance of a model's architecture, and encourages the combination of loss-function metalearning and neural architecture search.

8.4 Architectural Dependence

Experiments in the dissertation showed that some specific architectures work better with loss-function metalearning than others. Small changes, such as the different ordering of layers between ResNet and Preactivation ResNet architectures, can make TaylorGLO work better. Even changes to the depth and width of a particular architecture can affect TaylorGLO's performance.

Conceivably, different architectures have weight manifolds that are more amenable to regularization by loss functions. That is, loss functions can effect different spaces of changes to the manifold. These spaces of changes can have varying degrees of compatibility with the manifolds for different architectures, in a similar vein to how different architectures are more or less trainable themselves.

This hypothesis provides compelling motivation to coevolve loss functions and mutually beneficial architectures together, as discussed in Section 8.1. The analyses in Chapters 4, 5, and 7 were done on architectures that were designed with the cross-entropy loss, but perhaps there are other architectures that work better only with a different loss function.

Developing an understanding of how differences between architectures affect what performance improvements are attainable through loss-function metalearning is important future work. Such an understanding will allow better models to be trained and theoretical insights to be developed.

8.5 Prescription for AI Practitioners

The research presented in this dissertation opens up an exciting opportunity for AI researchers and engineers to enhance their models through loss-function evolution. Customized loss functions can improve a particular model's performance without humans needing to make hand-designed changes.

For most cases, TaylorGLO provides a practical approach that can be applied to existing deep neural networks with minimal changes. A TaylorGLO loss function can replace an existing loss function directly. The parameters to this loss function can then be evolved *in situ* for the task at hand.

Even if trained networks have comparable testing accuracies with TaylorGLO and the cross-entropy loss, networks trained with TaylorGLO loss functions are more robust against adversarial attacks. Similarly, any end-application objective—which does not need to be differentiable—can be used by TaylorGLO to evolve tailor-made loss functions that are customized to the needs of any particular application.

8.6 Broader Impact

While the main goal of the research in this dissertation is to improve deep learning, it is important to be cognizant that machine learning is a dual-use technology. Much as with other historically important fields and inventions, machine learning, and artificial intelligence more broadly, can usher positive societal impacts, but they can also make it possible for nefarious actors

to misappropriate it to harm humanity and the rights and liberties of individuals. Additionally, AI systems that are not properly supervised can result in unwanted outcomes. There is a natural tendency to trust such systems at face value if their perceived benefits are great enough. However, such trust can lead to scenarios where an AI system behaves in a counterintuitive manner, in a fashion that contradicts the wishes of humans, or in a domain in which it was not designed to function. There are also more indirect, negative impacts on climate, inequities, and fairness. It is important to take such concerns into account in the training, deployment, and use of AI systems.

8.6.1 Earth's Climate

As AI systems become larger and more complex, their power usage—which is correlated with compute usage—increases. In a world where a significant fraction of energy is still produced with polluting energy sources, there is a release of carbon dioxide associated with trained and deployed models that should not simply be ignored.

California, where the infrastructure that ran the experiments in this dissertation is located, is estimated to have had an estimated carbon dioxide equivalent total output emission rate of 226.21 $kgCO_2eq/kWh$ in 2018 [6]. This quantity can be used to calculate the climate impact of compute-intensive experiments.

Table 8.1 provides estimates of total emissions for various TaylorGLO experiments. They were calculated using the Machine Learning Impact cal-

Table 8.1: **Estimated total emissions resulting from individual TaylorGLO experiments with different configurations.** The estimates assume a population size of 20 and 50 generation runs. Values are upper bounds reported in equivalent kilograms of carbon dioxide, thus accounting for other gases of interest. Emission estimates show how the environmental impact of machine learning can vary greatly depending on the chosen model architecture. The impact is significant, and should be taken into account when planning experiments.

TaylorGLO Experiment	Total Emissions ($kgCO_2eq$)
AlexNet on CIFAR-10	4.07
ResNet-20 on CIFAR-10	11.58
Pre ResNet-20 on CIFAR-10	10.48
AllCNN-C on CIFAR-10	19.30
AllCNN-C + Aux. Classifiers on CIFAR-10	22.20
PyramidNet 110a48 on CIFAR-10	83.53
Wide ResNet 28-5 on CIFAR-10	48.52
Wide ResNet 16-8 on CIFAR-10	40.80
Wide ResNet 28-10 on CIFAR-10	119.10

culator [96], assuming that experiments were evenly distributed across both hardware configurations specified in Section 3.3.4, and that no candidates failed evaluation (which would result in slightly lower estimates). Presented values can be thought of as being an upper bound.

Machine learning research is not cheap: many of these estimates are on the same order-of-magnitude as the full-life-cycle carbon footprint of an iPhone 12 (i.e., 70 $kgCO_2eq$) [5]. Therefore, experiments should minimize compute usage, not only because it is costly, but because of its impact on the environment as well.

8.6.2 Research Community Inequities

Over the past decade, many key developments in deep learning in particular have risen out of large institutions with vast computational resources. Larger, more compute-intensive models tend to outperform smaller models. This trend has made it challenging for smaller groups to make contributions to the field. Evolution-based machine learning techniques, such as those embodied in this dissertation, inherently require large amounts of compute power. These techniques subsequently confer greater advantages to larger organizations than smaller ones. However, computers continually become more powerful, and thus many of these techniques will eventually be within reach of all researchers, in the same vein that the automobile or the smartphone were once exclusively used by those with greater economic means but are now accessible to more people.

8.6.3 Fairness, Safety, and Robustness

Fairness, safety, and robustness are becoming increasingly important as AI systems get deployed in the world, particularly when personal data is involved. Such systems have to operate in the real world, where data is not always as meticulously crafted as in a laboratory setting. AI systems impact people's lives directly, whether is in the form of a biased credit decision or an injury caused by an autonomous vehicle that misclassified a street sign. Thus, AI practitioners must be mindful of fairness, safety, and robustness throughout a system's development and deployment.

While fairness in training data is an entire issue in and of itself, the way in which a dataset is consumed can also affect whether the system is fair. Metalearned loss functions reduce overfitting in models, leading to models that tend to learn patterns rather than memorize individual samples. TaylorGLO, which was shown specifically to help guide learning away from overconfident memorization, could be a useful tool to help train fairer models.

TaylorGLO has also been shown to result in significantly more robust models, particularly in the face of adversarial attacks. This robustness has important implications for safety, where TaylorGLO models will less susceptible to out-of-distribution data and attacks by bad actors.

Therefore, while it is not a full solution, TaylorGLO can be used to make AI systems more fair, safe, and robust.

Chapter 9

Conclusion

This dissertation described the scientific journey that developed loss-function metalearning. This chapter closes this journey by reviewing the key contributions in this work. Closing remarks then reflect on how these contributions pushed the front line of knowledge further and how they can serve as a step in the further development of machine learning.

9.1 Contributions

This dissertation makes the following contributions to science:

Loss-Function Metalearning with GLO: Prior to this work, practitioners typically selected from a small number of loss functions for training their models. The metalearning of loss functions was first tackled in Chapter 4 with GLO, a novel technique to automatically discover new loss functions using evolution with a tree-based representation. Loss function trees can be further optimized through continuous coefficient evolution. The resulting loss functions were shown to outperform existing loss functions while training faster and performing well in low-data settings. Overall, GLO provides a flexible

approach that can be naturally extended, especially to domains where the loss function has a large number of inputs.

Efficient Loss-Function Metalearning with TaylorGLO: While the GLO approach provided unparalleled flexibility, it comes with a cost and is oftentimes unnecessary. TaylorGLO was introduced in Chapter 5 as an alternative technique with a more practical loss-function parameterization. Performance gains were demonstrated on many architectures across a variety of image classification benchmark datasets. Models trained with TaylorGLO loss functions were found to have very different training dynamics from those trained with the cross-entropy loss. These customized training dynamics allowed them to reach different parts of the trainable parameter space with flatter minima than standard loss functions, and thus better robutsness and generalization properties.

Theoretical Framework for Understanding Regularization: To understand how and why metalearned loss functions can train more performant networks, a theoretical framework for explaining loss-function regularization was developed in Chapter 6. A novel stochastic gradient descent learning rule decomposition allows any loss function to be decomposed into two expressions that describe their behavior. These behaviors were analyzed at the beginning of training and in a zero training error regime to elucidate the implicit biases that different loss functions impart. TaylorGLO loss functions are also found

to implicitly represent arbitrary degrees of label smoothing, another type of regularization, for any metalearned loss function, thus showing another dimension through metalearned loss functions can regularize networks. The entropy of the model's output distribution changes throughout the training process, illustrating how individual samples are treated by different loss functions, and demonstrating a key aspect of the regularizing behavior.

Inspiring Applications: The flexibility of loss-function metalearning with TaylorGLO was demonstrated in Chapter 7 through four unique applications. First, TaylorGLO was leveraged to evolve separate loss functions for models with auxiliary classifiers. TaylorGLO is able to take advantage of the different training dynamics associated with each auxiliary classifier to provide appropriate training signals at each point in the network. Second, using the theoretical framework in Chapter 6, an invariant on TaylorGLO parameters was integrated into the evolutionary search, resulting in further performance improvements. Third, TaylorGLO loss functions' robustness was leveraged to develop an approach to guard against adversarial attacks. TaylorGLO loss functions are already more resilient against such attacks compared to the cross-entropy loss; however an adversarial performance objective allowed it to discover new loss functions that improved robustness against adversarial attacks further. Fourth, TaylorGLO was applied to a unique type of model, conditional generative adversarial networks (GANs), where it discovered loss functions that lead to higher-quality generated images.

Practical Experimentation Platform: Chapter 3 introduced Lossferatu and Fumanchu, two components of a comprehensive distributed system that can manage, run, and analyze evolution-based experiments, such as those presented in this dissertation. The development of such a system was necessary to make extensive experiments possible. Lossferatu and Fumanchu were able to greatly speed up the pace of iteration and scientific discovery by removing much of the tedium, inefficiency, and manual effort that are typically found in roughly built research systems. Additionally, two libraries written in Swift, SwiftCMA and SwiftGenetics—which were used by Lossferatu—were open-sourced. SwiftCMA provides an implementation of CMA-ES, while SwiftGenetics is a representation-agnostic toolset that is used to build genetic algorithms through protocol-oriented programming.

9.2 Closing Remarks

Loss functions represent the fundamental objective that neural networks use to learn and thus greatly affect the performance of resulting models. In the abscense of a comprehensive theory for deep learning, optimal loss functions cannot be built from first principles; a search-based approach must be used.

Loss-function metalearning provides a new way to improve deep learning. By providing an automated way to improve neural network training, two new techniques—GLO and TaylorGLO—let practitioners train more robust models with better performance characteristics. These techniques discover

new loss functions that regularize by dynamically biasing the training process. The TaylorGLO technique in particular can be readily applied to modern deep neural network architectures with practically no manual tuning necessary.

The ability to optimize loss functions may make it possible to develop new classes of architectures that have been overlooked because they do not train well with traditional loss functions. Overall, loss-function metalearning is an important step in the journey towards models that are able to completely and automatically adapt to their target domain, resulting in the best possible performance.

Bibliography

[1] Kubernetes. `https://kubernetes.io`.

[2] MinIO. `https://min.io`.

[3] Nosferatu, 1922.

[4] Amazon Simple Storage Service. API reference, Amazon Web Services, Inc., Geneva, CH, March 2006.

[5] iPhone 12 Product Environmental Report. `https://www.apple.com/environment/pdf/products/iphone/iPhone_12_PER_Oct2020.pdf`, October 2020.

[6] United States Environmental Protection Agency eGRID2018. `https://www.epa.gov/egrid`, March 2020.

[7] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, Savannah, GA, 2016. USENIX Association.

[8] David Ackley and Michael Littman. Interactions between learning and evolution. *Artificial life II*, 10:487–509, 1991.

[9] Mohamed Baker Alawieh, Yibo Lin, Zaiwei Zhang, Meng Li, Qixing Huang, and David Z Pan. GAN-SRAF: Sub-resolution assist feature generation using conditional generative adversarial networks. In *Proceedings of the 56th Annual Design Automation Conference (DAC)*, page 149. ACM, 2019.

[10] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[11] Wolfgang Banzhaf, Peter Nordin, Robert E Keller, and Frank D Francone. *Genetic programming: An introduction*, volume 1. Morgan Kaufmann San Francisco, 1998.

[12] David Barber and Felix V. Agakov. Information maximization in noisy channels: A variational approach. In S. Thrun, L. K. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*, pages 201–208. MIT Press, 2004.

[13] Jonathan T Barron. A general and adaptive robust loss function. In

Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 4331–4339, 2019.

[14] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

[15] Yoshua Bengio, Samy Bengio, and Jocelyn Cloutier. *Learning a synaptic learning rule*. Université de Montréal, Département d'informatique et de recherche . . . , 1990.

[16] J. B. Biggs. The role of metalearning in study processes. *British Journal of Educational Psychology*, 55(3):185–212, 1985.

[17] Garrett Bingham, William Macke, and Risto Miikkulainen. Evolutionary optimization of deep learning activation functions. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2020.

[18] Christopher M Bishop. Regularization and complexity control in feed-forward networks. 1995.

[19] Guy Blanc, Neha Gupta, Gregory Valiant, and Paul Valiant. Implicit regularization for deep neural networks driven by an Ornstein-Uhlenbeck like process. In *Conference on Learning Theory*, pages 483–513, 2020.

[20] Ali Borji. Pros and cons of GAN evaluation measures. *Computer Vision and Image Understanding*, 179:41–65, 2019.

[21] Anna S. Bosman. *Fitness Landscape Analysis of Feed-Forward Neural Networks*. PhD thesis, University of Pretoria, 2019.

[22] Anna S. Bosman, Andries P. Engelbrecht, and Mardé Helbig. Progressive gradient walk for neural network fitness landscape analysis. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) Companion*, GECCO '18, pages 1473–1480, New York, NY, USA, 2018. ACM.

[23] Anna Sergeevna Bosman, Andries Engelbrecht, and Mardé Helbig. Visualising basins of attraction for the cross-entropy and the squared error neural network loss functions. *Neurocomputing*, 2020.

[24] Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-SGD: Biasing gradient descent into wide valleys. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124018, 2019.

[25] Pafnutiĭ L'vovich Chebyshev. *Théorie des mécanismes connus sous le nom de parallélogrammes*. Imprimerie de l'Académie impériale des sciences, 1853.

[26] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. In *Proceedings of the Fourth International Conference on Learning Representations (ICLR)*, 2016.

[27] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2172–2180. Curran Associates, Inc., 2016.

[28] JSR Chisholm. Rational approximants defined from double power series. *Mathematics of Computation*, 27(124):841–848, 1973.

[29] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical Japanese literature, 2018.

[30] Janet Clegg, James Alfred Walker, and Julian Frances Miller. A new crossover technique for Cartesian genetic programming. In *Proceedings of the 9th Genetic and Evolutionary Computation Conference (GECCO)*, pages 1580–1587. ACM, 2007.

[31] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 113–123, 2019.

[32] Charles Darwin. *On the Origin of Species*. 1859.

[33] Charles Darwin and Alfred Wallace. On the tendency of species to form varieties; and on the perpetuation of varieties and species by natural means of selection. *Journal of the proceedings of the Linnean Society of London. Zoology*, 3(9):45–62, 1858.

[34] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[35] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with Cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[36] Hao Dong, Simiao Yu, Chao Wu, and Yike Guo. Semantic image synthesis via adversarial learning. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 5706–5714, 2017.

[37] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[38] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*,

20(55):1–21, 2019.

[39] William Fedus, Mihaela Rosca, Balaji Lakshminarayanan, Andrew M. Dai, Shakir Mohamed, and Ian Goodfellow. Many paths to equilibrium: GANs do not need to decrease a divergence at every step. In *Proceedings of the Sixth International Conference on Learning Representations (ICLR)*, 2018.

[40] Richard Forsyth. BEAGLE — a Darwinian approach to pattern recognition. *Kybernetes*, 10(3):159–166, 1981.

[41] Joseph BJ Fourier. La théorie analytique de la chaleur. *Mémoires de l'Académie Royale des Sciences de l'Institut de France*, 8:581–622, 1829.

[42] AS Fraser. Monte Carlo analyses of genetic models. *Nature*, 181(4603):208, 1958.

[43] Maurice Fréchet. Sur la distance de deux lois de probabilité. *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences*, 244(6):689–692, 1957.

[44] Adam Gaier and David Ha. Weight agnostic neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, volume 32, pages 5364–5378. Curran Associates, Inc., 2019.

[45] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of*

the 33rd International Conference on Machine Learning (ICML), pages 1050–1059, 2016.

[46] Angus Galloway, Anna Golubeva, Thomas Tanay, Medhat Moussa, and Graham W Taylor. Batch normalization is a cause of adversarial vulnerability. In *Seventh International Conference on Learning Representations (ICLR), Deep Phenomena Workshop*, 2019.

[47] Ruohan Gao and Kristen Grauman. 2.5D visual sound. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 324–333, 2019.

[48] Aditya Sharad Golatkar, Alessandro Achille, and Stefano Soatto. Time matters in regularizing deep networks: Weight decay and data augmentation affect early learning dynamics, matter little near convergence. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, volume 32, pages 10678–10688. Curran Associates, Inc., 2019.

[49] Santiago Gonzalez. SwiftCMA. `https://github.com/sgonzalez/SwiftCMA`.

[50] Santiago Gonzalez. SwiftGenetics. `https://github.com/sgonzalez/SwiftGenetics`.

[51] Santiago Gonzalez, Joshua Landgraf, and Risto Miikkulainen. Faster training by selecting samples using embeddings. In *2019 International*

Joint Conference on Neural Networks (IJCNN), 2019.

[52] Santiago Gonzalez and Risto Miikkulainen. Improved training speed, accuracy, and data utilization through loss function optimization. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2020.

[53] Santiago Gonzalez and Risto Miikkulainen. Improved training speed, accuracy, and data utilization through loss function optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2020.

[54] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[55] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[56] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *Proceedings of the Third International Conference on Learning Representations (ICLR)*, 2015.

[57] "Student" William Sealy Gosset. The probable error of a mean. *Biometrika*, pages 1–25, 1908.

[58] PR Graves-Morris. The numerical calculation of Padé approximants. In *Padé approximation and its applications*, pages 231–245. Springer, 1979.

[59] PR Graves-Morris and DE Roberts. Calculation of Canterbury approximants. *Computer Physics Communications*, 10(4):234–244, 1975.

[60] John J Grefenstette and J Michael Fitzpatrick. Genetic search with approximate function evaluations. In *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, pages 112–120, 1985.

[61] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of Wasserstein GANs. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5767–5777. Curran Associates, Inc., 2017.

[62] Ishaan Gulrajani, Colin Raffel, and Luke Metz. Towards GAN benchmarks which require generalization. In *Proceedings of the Seventh International Conference on Learning Representations (ICLR)*, 2019.

[63] Dongyoon Han, Jiwhan Kim, and Junmo Kim. Deep pyramidal residual networks. In *Proceedings of the IEEE Conference on Computer Vision*

and Pattern Recognition (CVPR), pages 5927–5935, 2017.

[64] Nikolaus Hansen and Stefan Kern. Evaluating the CMA evolution strategy on multimodal test functions. In *International Conference on Parallel Problem Solving from Nature*, pages 282–291. Springer, 2004.

[65] Nikolaus Hansen and Andreas Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proceedings of IEEE international conference on evolutionary computation*, pages 312–317. IEEE, 1996.

[66] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[67] Stephen Hanson and Lorien Pratt. Comparing biases for minimal network construction with back-propagation. In D. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, volume 1, pages 177–185. Morgan-Kaufmann, 1989.

[68] Moritz Hardt, Ben Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, pages 1225–1234, 2016.

[69] Jacob Harer, Onur Ozdemir, Tomo Lazovich, Christopher Reale, Rebecca Russell, Louis Kim, and peter chin. Learning to repair soft-

ware vulnerabilities with generative adversarial networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 7933–7943. Curran Associates, Inc., 2018.

[70] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[71] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[72] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision (ECCV)*, pages 630–645. Springer, 2016.

[73] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, volume 30, pages 6626–6637. Curran Associates, Inc., 2017.

[74] Geoffrey E Hinton and Terrence J Sejnowski. Optimal perceptual inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 448–453. Citeseer, 1983.

[75] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[76] John Henry Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. AUniversity of Michigan Press, 1975.

[77] Rein Houthooft, Yuhua Chen, Phillip Isola, Bradly Stadie, Filip Wolski, OpenAI Jonathan Ho, and Pieter Abbeel. Evolved policy gradients. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, volume 31, pages 5400–5409. Curran Associates, Inc., 2018.

[78] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4700–4708, 2017.

[79] Xun Huang, Yixuan Li, Omid Poursaeed, John Hopcroft, and Serge Belongie. Stacked generative adversarial networks. In *Proceedings*

of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017.

[80] Peter J Huber. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, pages 73–101, 1964.

[81] Apple Inc. libdispatch. `https://apple.github.io/swift-corelibs-libdispatch`.

[82] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 448–456, 2015.

[83] Takashi Ishida, Ikko Yamane, Tomoya Sakai, Gang Niu, and Masashi Sugiyama. Do we need zero training loss after achieving zero training error? *arXiv preprint arXiv:2002.08709*, 2020.

[84] Information technology – Advanced Message Queuing Protocol (AMQP) v1.0 specification. Standard, International Organization for Standardization, Geneva, CH, May 2000.

[85] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *arxiv*, 2016.

[86] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

[87] Yaochu Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1:61–70, 06 2011.

[88] Dimitris Kalimeris, Gal Kaplun, Preetum Nakkiran, Benjamin Edelman, Tristan Yang, Boaz Barak, and Haofeng Zhang. SGD on neural networks learns functions of increasing complexity. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 3496–3506. Curran Associates, Inc., 2019.

[89] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. In *Proceedings of the Sixth International Conference on Learning Representations (ICLR)*, 2018.

[90] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *Proceedings of the Fifth International Conference on Learning Representations (ICLR)*, 2017.

[91] Diederik Kingma and Max Welling. Auto-encoding variational Bayes. In *Proceedings of the Second International Conference on Learning Representations (ICLR)*, 12 2014.

[92] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[93] John R Koza. Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines*, 11(3-4):251–284, 2010.

[94] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[95] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[96] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*, 2019.

[97] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. *arXiv preprint arXiv:1512.09300*, 2015.

[98] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[99] Yann LeCun, Corinna Cortes, and CJC Burges. The MNIST dataset of handwritten digits, 1998.

[100] Joel Lehman et al. The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities. *arXiv preprint arXiv:1803.03453*, 2018.

[101] Christiane Lemke, Marcin Budka, and Bogdan Gabrys. Metalearning: a survey of trends and technologies. *Artificial Intelligence Review*, 44(1):117–130, 2015.

[102] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6389–6399. Curran Associates, Inc., 2018.

[103] Ming Li, Rui Xi, Beier Chen, Mengshu Hou, Daibo Liu, and Lei Guo. Generate desired images from trained generative adversarial networks. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.

[104] Yuanzhi Li, Colin Wei, and Tengyu Ma. Towards explaining the regularization effect of initial large learning rate in training neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 11674–11685. Curran Associates, Inc., 2019.

[105] Jason Liang, Elliot Meyerson, Babak Hodjat, Dan Fink, Karl Mutch, and Risto Miikkulainen. Evolutionary neural autoML for deep learning.

In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 401–409, 2019.

[106] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 3730–3738, 2015.

[107] Jason D Lohn, Gregory S Hornby, and Derek S Linden. Evolution, re-evolution, and prototype of an X-band antenna for NASA's Space Technology 5 mission. In *Proceedings of the International Conference on Evolvable Systems*, pages 205–214. Springer, 2005.

[108] Ilya Loshchilov and Frank Hutter. CMA-ES for hyperparameter optimization of deep neural networks. *arXiv preprint arXiv:1604.07269*, 2016.

[109] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *Proceedings of the Sixth International Conference on Learning Representations (ICLR)*, 2018.

[110] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[111] Xudong Mao, Qing Li, Haoran Xie, Raymond Yiu Keung Lau, Zhen Wang, and Stephen Paul Smolley. On the effectiveness of least squares generative adversarial networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

[112] Xudong Mao, Qing Li, Haoran Xie, Raymond Y.K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[113] Tim McGraw. *Live Like You Were Dying*. Jan 2004.

[114] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163*, 2016.

[115] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293–312. Elsevier, 2019.

[116] Julian F Miller, Peter Thomson, and Terence Fogarty. Designing electronic circuits using evolutionary algorithms. arithmetic circuits: A case study, 1997.

[117] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[118] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *Proceedings of the Sixth International Conference on Learning Representations (ICLR)*, 2018.

[119] N. Morgan and H. Bourlard. Generalization and parameter estimation in feedforward nets: Some experiments. In D. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, volume 2, pages 630–637. Morgan-Kaufmann, 1990.

[120] Karl Mutch. Studio Go Runner. `https://hub.docker.com/repository/docker/leafai/studio-go-runner`, 2017 - 2020.

[121] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 807–814, 2010.

[122] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. In *Proceedings of the Seventh International Conference on Learning Representations (ICLR)*, 2019.

[123] John Nash. Non-cooperative games. *Annals of Mathematics*, pages 286–295, 1951.

[124] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. *Neural Information Processing Systems 24, Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[125] Behnam Neyshabur, Srinadh Bhojanapalli, and Nathan Srebro. A PAC-Bayesian approach to spectrally-normalized margin bounds for neural

networks. In *Proceedings of the Sixth International Conference on Learning Representations (ICLR)*, 2018.

[126] Behnam Neyshabur, Russ R Salakhutdinov, and Nati Srebro. Path-SGD: Path-normalized optimization in deep neural networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2422–2430. Curran Associates, Inc., 2015.

[127] Behnam Neyshabur, Ryota Tomioka, Ruslan Salakhutdinov, and Nathan Srebro. Data-dependent path normalization in neural networks. *arXiv preprint arXiv:1511.06747*, 2015.

[128] Behnam Neyshabur, Ryota Tomioka, Ruslan Salakhutdinov, and Nathan Srebro. Geometry of optimization and implicit regularization in deep learning. *arXiv preprint arXiv:1705.03071*, 2017.

[129] Scott Niekum, Andrew G Barto, and Lee Spector. Genetic programming for reward function search. *IEEE Transactions on Autonomous Mental Development*, 2(2):83–90, 2010.

[130] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016.

[131] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary

DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[132] Karl Pearson. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 50(302):157–175, 1900.

[133] Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. Regularizing neural networks by penalizing confident output distributions. In *Fifth International Conference on Learning Representations (ICLR), Workshop paper*, 2017.

[134] Xin Qiu, Elliot Meyerson, and Risto Miikkulainen. Quantifying point-prediction uncertainty in neural networks via residual estimation with an I/O kernel. In *Proceedings of the Seventh International Conference on Learning Representations (ICLR)*, 2019.

[135] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[136] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4780–4789, 2019.

[137] Esteban Real, Chen Liang, David R. So, and Quoc V. Le. AutoML-Zero: Evolving machine learning algorithms from scratch. *arXiv:2003.03384*, 2020.

[138] Ingo Rechenberg. Evolutionsstrategien. In *Simulationsmethoden in der Medizin und Biologie*, pages 83–114. Springer, 1978.

[139] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, volume 48 of *Proceedings of Machine Learning Research*, pages 1060–1069, New York, New York, USA, 20–22 Jun 2016. PMLR.

[140] Wolfram Research, Inc. Mathematica, Version 12.1. Champaign, IL, 2020.

[141] John R Rice. The algorithm selection problem. In *Advances in Computers*, volume 15, pages 65–118. Elsevier, 1976.

[142] Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab

Inc Buffalo NY, 1961.

[143] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[144] Carl Runge. Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten. *Zeitschrift für Mathematik und Physik*, 46(224-243):20, 1901.

[145] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[146] Graeme D Ruxton. The unequal variance t-test is an underused alternative to Student's t-test and the Mann–Whitney U test. *Behavioral Ecology*, 17(4):688–690, 2006.

[147] Levent Sagun, Utku Evci, V Ugur Guney, Yann Dauphin, and Leon Bottou. Empirical analysis of the Hessian of over-parametrized neural networks. *arXiv preprint arXiv:1706.04454*, 2017.

[148] Levent Sagun, Utku Evci, Veli Uğur Güney, Yann Dauphin, and Léon Bottou. Empirical analysis of the hessian of over-parametrized neural

networks. In *Sixth International Conference on Learning Representations (ICLR), Workshop paper*, 2018.

[149] Ruslan Salakhutdinov and Geoffrey Hinton. Deep Boltzmann machines. In *Artificial Intelligence and Statistics*, pages 448–455, 2009.

[150] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training GANs. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, volume 29, pages 2234–2242. Curran Associates, Inc., 2016.

[151] Franklin E Satterthwaite. An approximate distribution of estimates of variance components. *Biometrics bulletin*, 2(6):110–114, 1946.

[152] Pedro Savarese, Itay Evron, Daniel Soudry, and Nathan Srebro. How do infinite width bounded norm networks look in function space? In Alina Beygelzimer and Daniel Hsu, editors, *Proceedings of the Thirty-Second Conference on Learning Theory*, volume 99 of *Proceedings of Machine Learning Research*, pages 2667–2690, Phoenix, USA, 25–28 Jun 2019. PMLR.

[153] Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

[154] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[155] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

[156] Hormoz Shahrzad, Daniel Fink, and Risto Miikkulainen. Enhanced optimization with composite objectives and novelty selection. In *Artificial Life Conference Proceedings*, pages 616–622. MIT Press, 2018.

[157] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[158] Leslie N Smith. Cyclical learning rates for training neural networks. In *Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472. IEEE, 2017.

[159] Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Technical report, Colorado University at Boulder Deptartment of Computer Science, 1986.

[160] Lee Spector, Erik Goodman, A Wu, W B. Langdon, H m. Voigt, M Gen, S Sen, M Dorigo, S Pezeshk, M Garzon, E Burke, and Morgan Kaufmann Publishers. Autoconstructive evolution: Push, PushGP, and Pushpop. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 05 2001.

[161] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2015.

[162] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research JMLR*, 15(1):1929–1958, 2014.

[163] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. In *32nd International Conference on Machine Learning (ICML), Deep Learning Workshop*, 2015.

[164] Kenneth O. Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35, 2019.

[165] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[166] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[167] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the Inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

[168] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *Proceedings of the First International Conference on Learning Representations (ICLR)*, 2013.

[169] Brook Taylor. *Methodus incrementorum directa & inversa. Auctore Brook Taylor, LL. D. & Regiae Societatis Secretario.* typis Pearsonianis: prostant apud Gul. Innys ad Insignia Principis in . . . , 1715.

[170] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the condor experience. *Concurrency and computation: practice and experience*, 17(2-4):323–356, 2005.

[171] L Theis, A van den Oord, and M Bethge. A note on the evaluation of generative models. In *Proceedings of the Fourth International Conference on Learning Representations (ICLR)*, pages 1–10, 2016.

[172] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[173] Andrey N. Tikhonov. Solution of incorrectly formulated problems and the regularization method. In *Proceedings of the USSR Academy of Sciences*, volume 4, pages 1035–1038, 1963.

[174] A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

[175] Alan Mathison Turing. Intelligent machinery, 1948.

[176] Radim Tyleček and Radim Šára. Spatial pattern templates for recognition of objects with regular structure. In *Proceeding of the German Conference on Pattern Recognition (GCPR)*, pages 364–374, Saarbrucken, Germany, 2013. Springer.

[177] `https://github.com/rabbitmq/rabbitmq-server/graphs/contributors`. RabbitMQ. `https://rabbitmq.com`.

[178] `https://github.com/studioml/studio/graphs/contributors`. Studio. `https://studio.ml`, 2017 - 2020.

[179] Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Koray Kavukcuoglu, Oriol Vinyals, and Alex Graves. Conditional image generation with PixelCNN decoders. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4790–4798. Curran Associates, Inc., 2016.

[180] David A Van Dyk and Xiao-Li Meng. The art of data augmentation. *Journal of Computational and Graphical Statistics*, 10(1):1–50, 2001.

[181] Leonid Nisonovich Vaserstein. Markov processes over denumerable products of spaces, describing large systems of automata. *Problemy Peredachi Informatsii*, 5(3):64–72, 1969.

[182] Cédric Villani. The Wasserstein distances. In *Optimal Transport*, pages 93–111. Springer, 2009.

[183] Vanessa Volz, Jacob Schrum, Jialin Liu, Simon M Lucas, Adam Smith, and Sebastian Risi. Evolving Mario levels in the latent space of a deep convolutional generative adversarial network. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 221–228. ACM, 2018.

[184] Stefan Wager, Sida Wang, and Percy S Liang. Dropout training as adaptive regularization. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, volume 26, pages 351–359. Curran Associates, Inc., 2013.

[185] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional GANs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8798–8807, 2018.

[186] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error measurement to structural similarity.

IEEE Transactions on Image Processing, 13(1), 2004.

[187] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 2, pages 1398–1402. IEEE, 2003.

[188] Edward Waring. Problems concerning interpolations. *Philosophical transactions of the royal society of London*, (69):59–67, 1779.

[189] Bernard L Welch. The generalization of Student's problem when several different population variances are involved. *Biometrika*, 34(1/2):28–35, 1947.

[190] Henry Wilbraham. On a certain periodic function. *The Cambridge and Dublin Mathematical Journal*, 3:198–201, 1848.

[191] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4148–4158. Curran Associates, Inc., 2017.

[192] David H Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

[193] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 82–90. Curran Associates, Inc., 2016.

[194] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms, 2017.

[195] Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. Exploring randomly wired neural networks for image recognition. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1284–1293, 2019.

[196] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315, 2007.

[197] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. CutMix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 6023–6032, 2019.

[198] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[199] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *Proceedings of the Sixth International Conference on Learning Representations (ICLR)*, 2018.

[200] Yao Zhou, Cong Liu, and Yan Pan. Modelling sentence pairs with tree-structured attentive encoder. In *Proceedings of the 26th International Conference on Computational Linguistics (COLING), Technical Papers*, pages 2912–2922, 2016.

[201] Zhiming Zhou, Han Cai, Shu Rong, Yuxuan Song, Kan Ren, Weinan Zhang, Yong Yu, and Jun Wang. Activation maximization generative adversarial nets. *arXiv preprint arXiv:1703.02000*, 2017.

[202] Donald W Zimmerman. A note on preliminary tests of equality of variances. *British Journal of Mathematical and Statistical Psychology*, 57(1):173–181, 2004.

[203] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8697–8710, 2018.

Vita

Santiago Gonzalez was born in Mexico City, Mexico in 1998. He received the Bachelor of Science and Master of Science degrees in Computer Science from the Colorado School of Mines in December of 2014 and 2015, respectively. He started pursuing his Doctorate of Philosophy in Computer Science at the University of Texas at Austin in August of 2016.

Permanent address: slgonzalez@utexas.edu

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.