Advancements in Efficient Training Strategies for Modern Deep Learning: From Implicit Deep Learning to Language Models and Beyond

Tanmay Gautam

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2024-20
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2024/EECS-2024-20.html

April 25, 2024

Advancements in Efficient Training Strategies for Modern Deep Learning: From Implicit Deep Learning to Language Models and Beyond

by

Tanmay Gautam

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Somayeh Sojoudi, Chair
Professor Laurent El Ghaoui
Professor Javad Lavaei

Spring 2024

Advancements in Efficient Training Strategies for Modern Deep Learning: From Implicit Deep Learning to Language Models and Beyond

Abstract

Advancements in Efficient Training Strategies for Modern Deep Learning: From Implicit Deep Learning to Language Models and Beyond

by

Tanmay Gautam

Doctor of Philosophy in Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Somayeh Sojoudi, Chair

In the rapidly evolving landscape of machine learning, the surge in computing power and data has propelled deep learning to the forefront of academic research. As the scale of models and datasets continues to expand, an increasing emphasis is placed on algorithmic enhancements to tackle the growing compute and memory requirements. Moreover, owing to its success across a wide range of applications, the domain has seen a proliferation of diverse neural network architectures each with their own unique training challenges. This thesis introduces efficient training methods for prevalent neural network architectures that leverage model structure for both resource and algorithmic efficiency.

In the first part, we first present novel training algorithms with reduced computational and memory demands for implicit deep learning models and transformer-based language models. Specifically, we start by proposing an efficient sequential training method for implicit equilibrium models, which eliminates the need to solve computationally expensive fixed-point equations and projection steps within the existing training process. We then introduce variance-reduced zeroth-order methods to effectively fine-tune large language models using only memory-efficient inference passes.

In the second part, we shift our focus to exploring the application of differentiable optimization to enhance training within meta-optimization and vector quantization. Specifically, for the former, we propose a means to use structure presented by differentiable convex optimization to parameterize novel first-order optimizers. For the latter, we introduce differentiable convex optimization as a technique to improve backpropagation through vector quantization layers.

We hope that this work will offer fresh viewpoints to the research community and serve as a foundation to further develop efficient training strategies for deep learning.

To my parents,
who have been the anchor throughout my life's journey.

Contents

Contents **ii**

1 Introduction **1**

I Efficient Training Strategies for Deep Neural Architectures **4**

2 Sequential Training for Implicit Deep Models **5**
2.1 Introduction . . . 5
2.2 Background . . . 8
2.3 Strictly Block Triangular Implicit Models . . . 9
2.4 Sequential Blockwise Training . . . 11
2.5 Conclusion . . . 16
2.6 Supplementary Information: Generalization of Theorem 2 . . . 17

3 Zeroth-Order Methods for Fine-tuning Language Models **21**
3.1 Introduction . . . 21
3.2 Background . . . 22
3.3 Our proposed method: MeZO-SVRG . . . 25
3.4 Experiments . . . 29
3.5 Convergence Theory . . . 32
3.6 Related work . . . 33
3.7 Conclusion . . . 34
3.8 Additional Results: Proof of Theorem . . . 35
3.9 Additional Results: Exploring the Limits of MeZO Empirically . . . 36
3.10 Additional Results: Zeroth-Order Stochastic Variance-Reduced Gradient . . . 38
3.11 Additional Results: Experiment Setup . . . 39
3.12 Additional Results: MeZO-SVRG Implementation and Ablations . . . 40
3.13 Additional Results: Fine-tuning DistilBert . . . 45
3.14 Additional Results: Fine-tuning RoBERTa-large . . . 46
3.15 Additional Results: Additional Results for fine-tuning Autoregressive Models . . . 47
3.16 Additional Results: Memory Usage and Computation Time . . . 49

3.17 Additional Results: Half-Precision Experiments 53

II Enhancing Training with Differentiable Optimization 58

4 Meta-Learning Parameterized First-Order Optimizers 59
4.1 Introduction 59
4.2 Background 62
4.3 Meta-Optimization Framework 63
4.4 Differentiable Convex Optimizers 67
4.5 Theory 68
4.6 Experiments 71
4.7 Conclusion 74

5 Revisiting Vector Quantization via Convex Optimization 75
5.1 Introduction 75
5.2 Background 77
5.3 Methodology 79
5.4 Experiments 82
5.5 Conclusion 85
5.6 Additional Results: VQVAE Experiments 86
5.7 Additional Results: VQGAN Experiments 88

6 Conclusion 90

Bibliography 91

Acknowledgments

Firstly, I would like to extend my deepest gratitude to my PhD advisor, Professor Somayeh Sojoudi, alongside my dissertation and qualifying exam committee members, Professors Laurent El Ghaoui, Javad Lavaei and Murat Arcak, for your unwavering support and invaluable guidance throughout my academic journey. Your contributions have been indispensable, and for that, I am profoundly thankful.

I am a firm believer that collaboration and an open exchange ideas is pivotal to the pursuit of meaningful research. To this end, my PhD journey has been immensely enriched by fruitful collaborations both during internships within industry as well as in my academic research lab. I extend my heartfelt gratitude to my esteemed colleagues and mentors at Microsoft—Chi Wang, Wentao Wu, Reid Pryzant, Ziyi Yang, and Chenguang Zhu—and at Amazon—Youngsuk Park, Wooseok Ha, Gaurav Gupta, Parameswaran Raman, Hao Zhou, and Luke Huan. Your support and continuous guidance throughout our research projects are greatly appreciated. The fervor of our discussions and your insightful contributions have significantly shaped my evolution as a researcher.

Similarly, I would like to extend my sincerest gratitude to my PhD colleagues - Sam Pfrommer, Brendon Anderson, Yatong Bai, Ziye Ma, Hyunin Lee, Eli Brock and Jingqi Li. You made the rigors of a PhD journey seem far more enjoyable than I could have possibly imagined: thank you for the wonderful time during conference travel, your feedback during lab meetings and the fun discussions during our lunch hangouts.

Like life, pursuing a PhD is about finding balance: for every stressful deadline and project milestone, there must be moments to unwind. A crucial part of my relaxation was spending quality time with the many friends I had the pleasure of meeting along my PhD journey. Thank you, Sam, Alonso, Kartik, Nitya, Tanmay V, Shafeeq, and Jerome, for the countless evenings filled with dance, dinners, tennis, and adventures around the Bay Area. And a special thanks to Kartik and Nitya for being wonderful flatmates and travel companions.

My heartfelt appreciation is dedicated to my parents - without whom my journey here would not have been possible. Words fall short of capturing the depth of your endless sacrifices. This PhD stands as a testament to your unconditional love and support!

Chapter 1

Introduction

The past decade has witnessed unprecedented advancements in the field of artificial intelligence (AI) that have culminated in superhuman performance across a wide array of specialized tasks within domains such as natural language processing and computer vision. The strides in AI have been facilitated by a synergy of architectural innovation and computational improvements within deep learning [1], [2].

Until recently, research efforts within deep learning were typically specialized, focusing on particular domains such as NLP or vision. In each application area, research aimed towards developing tailored neural network architectures designed to address the specific challenges of the application. For example, recurrent neural networks (RNNs) and their variants were used to process sequential data typically seen within NLP. On the other hand, vision applications often employed Convolutional Neural Networks due to their ability to efficiently handle visual data. The specialization was seen as necessary, because different data modalities required tailored processing methodologies to learn the underlying patterns. This spurred on a proliferation of architecture types across domains.

Recently, the introduction of transformers and implicit deep learning have brought about a departure from developing domain-specific architectures. Transformer models are built upon the notion of attention: a mechanism that accommodates long-term dependencies within sequential data, enables parallel processing and remains compatible with backpropagation. In particular, transformer-based architectures are now ubiquitous in state-of-the-art models, setting performance benchmarks across both NLP and vision tasks. Implicit deep learning moves away from the notion of neural networks as a stack of explicit, feed-forward layers and represents them implicitly through a set of conditions that the output should satisfy. This paradigm offers an expressive model class with several instantiations including Neural Ordinary Differential Equations, differentiable optimization and Deep Equilibrium models. Concretely, in [3], implicit models are shown to generalize most of the popular deep learning architectures with promising performance across a wide array of illustrative applications.

The emergence of new, expressive deep learning architectures has emphasized the significance of developing efficient optimization strategies to unlock their complete performance potential. More concretely, developing optimization strategies tailored towards different architecture-types is foundational to efficient model training that learns effectively from data. This underscores the

need for continual refinement of training techniques together with architecture design to fully realize the potential of deep learning technologies.

This thesis contributes to a broader effort of developing tailored training strategies to address the unique needs of state-of-the-art deep learning architectures. In Part I, we begin by examining the resource-intensive nature of existing training methods for implicit deep learning and transformer models, and propose novel algorithms to overcome the prohibitive compute and memory demands. In Part II our focus changes to exploring how the particular implicit deep learning instantiation of differentiable optimization can be leveraged as a technique to enhance the training process within meta-optimization and vector quantization.

Part I: Efficient Training Strategies for Deep Neural Architectures

In this part, we draw attention to training challenges faced by prevalent architectural types and propose optimization algorithms designed to mitigate each of the specific challenges. Specifically, we aim to overcome the prohibitive compute and memory requirements of existing training methods for implicit deep learning and transformer-based language models.

In Chapter 2, we highlight the drawbacks of the existing training method for implicit models characterized via fixed-point equations: an end-to-end optimization scheme that leverages computationally cumbersome implicit differentiation and projection steps. We propose a novel sequential, blockwise training algorithm for upper triangular implicit deep models that mitigates the need for implicit differentiation and projection steps.

In Chapter 3, we address the large memory requirements of first-order methods when fine-tuning transformer-based language models (LMs). Based on the observation that Zeroth-Order (ZO) methods estimate gradients using only memory-efficient inference passes, we couple ZO methods with variance reduction techniques to enhance stability and convergence for inference-based LM fine-tuning. Our experiments demonstrate consistently improved performance over existing ZO fine-tuning benchmarks whilst retaining a significantly lower memory-footprint compared to first-order methods.

Part II: Enhancing Training with Differentiable Optimization

In Part II, we focus on the application of differentiable optimization as a means to improve the learning process of meta-optimization and vector quantization.

In Chapter 4, we demonstrate how convex optimization can be used to generalize many existing first-order update rules. We then propose a new data-driven approach for optimization algorithm design that leverages differentiable convex optimization (DCO). Using this previous optimization experience can be used to propose novel update rules that efficiently solve new optimization tasks sampled from the same underlying problem class. We demonstrate on illustrative experiments that DCO optimizers are able to outperform prevalent first-order methods that are used in practice.

In Chapter 5, we leverage DCO to mitigate the training challenges posed by vector quantization (VQ) layers. VQ-embedded models have shown impressive results in a range of applications including image and speech generation. VQ operates as a parametric K-means algorithm that quan-

tizes inputs using a single codebook vector in the forward pass. While powerful, this technique faces practical challenges including codebook collapse, non-differentiability and lossy compression. To mitigate the aforementioned issues, we propose Soft Convex Quantization (SCQ) as a direct substitute for VQ. SCQ works like a differentiable convex optimization (DCO) layer: in the forward pass, we solve for the optimal convex combination of codebook vectors that quantize the inputs. In the backward pass, we leverage differentiability through the optimality conditions of the forward solution. We then introduce a scalable relaxation of the SCQ optimization and demonstrate its efficacy on the CIFAR-10 [4], GTSRB [5] and LSUN [6] datasets. We train powerful SCQ autoencoder models that significantly outperform matched VQ-based architectures, observing an order of magnitude better image reconstruction and codebook usage with comparable quantization runtime.

Part I

Efficient Training Strategies for Deep Neural Architectures

Chapter 2

Sequential Training for Implicit Deep Models

2.1 Introduction

Over the past decades, the rise in computing power and available data has propelled deep learning to the forefront of machine learning research [1]. Deep learning has also found many applications in control and decision problems, including estimation for power systems, control of autonomous vehicles, and reinforcement learning for systems with uncertainty [7]–[9]. Current deep learning models are built upon the notion of hierarchical architectures, where input information is processed recursively through several forward-propagating, differentiable parametric layers [1]. Canonical examples of this type are standard feed-forward neural networks (FFNs) and convolutional neural networks (CNNs) commonly used to perform image classification [10], [11].

Recent work has proposed a new perspective wherein deep learning models can be analyzed based on implicit prediction rules [3]. This framework, termed "implicit deep learning," is based on a model consisting of a prediction equation and a fixed-point equilibrium equation in a state vector $x \in \mathbb{R}^n$:

$$\begin{aligned} \hat{y}(u) &= \mathbf{C}x + \mathbf{D}u, && \text{(prediction equation)} \\ x &= \phi(\mathbf{A}x + \mathbf{B}u), && \text{(fixed-point equation)} \end{aligned}$$

where $\phi\colon \mathbb{R}^n \to \mathbb{R}^n$ is a nonlinear activation map, $\mathbf{A} \in \mathbb{R}^{n\times n}$, $\mathbf{B} \in \mathbb{R}^{n\times p}$, $\mathbf{C} \in \mathbb{R}^{q\times n}$, and $\mathbf{D} \in \mathbb{R}^{q\times p}$ are model parameters, and where $u \in \mathbb{R}^p$ and $\hat{y} \in \mathbb{R}^q$ are respectively the input and output. The fixed-point equation can be readily interpreted as a discrete-time linear time-invariant (LTI) system composed with the nonlinearity ϕ evaluated at the system's equilibrium. The prediction equation is equivalent to the standard output equation seen in LTI systems.

The significance of implicit deep learning lies in its representational power: [3] illustrates how it encapsulates most of the current neural network architectures as special cases, including feed-forward, convolutional, and residual networks. This framework can be regarded as a more general

model that offers greater capacity to possibly model novel prediction rules for deep learning that may not necessarily be tied to any notion of "network" or "layers" [3].

Previously, the training of implicit models has relied on an end-to-end optimization approach via (stochastic) projected gradient methods, where the gradient is found by implicitly differentiating through the fixed-point equation [3]. This end-to-end approach for dense implicit models suffers from some notable drawbacks. First, computing the gradient at each training step is non-trivial, as it requires solving a separate fixed-point equation in a matrix variable. The gradient step must be followed by a computationally cumbersome projection of the matrix variable $\mathbf{A}$ to ensure well-posedness [3]. Furthermore, optimizing over all model parameters with a global objective obscures the interpretability of the model, as we cannot gain insight into the contribution of each subset of the parameters. Finally, it is known from conventional deep learning that end-to-end optimization landscapes are highly chaotic when training large models [12].

Contributions

In this chapter, we focus on the efficient training of implicit models with a strictly upper block triangular structure. We first motivate imposing this structure by relating strictly upper block triangular implicit models to generalized dense block modules that constitute Dense Convolutional Networks (DenseNets) and in turn highlight some of the strengths of implicit models with the aforementioned structure via the advantages of DenseNets. Subsequently, we propose a novel sequential, greedy, blockwise training algorithm for implicit deep models with the assumed structure. Unlike end-to-end training, the approach is interpretable and does not rely on projection steps or implicit differentiation. We posit that this decomposition into blockwise training alleviates the chaotic optimization landscapes that arise when performing end-to-end training on large implicit models. We experimentally show that our sequential approach outperforms end-to-end training, which aligns with prior findings in the conventional deep learning literature claiming that end-to-end optimization is susceptible to poor local solutions. Finally, we theoretically prove that, for ReLU implicit models, the training for more general, non-strictly upper triangular models (with self-loops between feature blocks) is equivalent to the special case of strictly upper block triangular training, a result novel to the literature.

Related Works

Implicit Deep Learning Implicit deep learning is largely inspired by the concurrent works [3], [13]. In [3], implicit models are shown to generalize most of the popular deep learning architectures, sufficient conditions are proven for the uniqueness of their predictions, and methods to assess their robustness are proposed. The paper [13] shows that implicit models are equivalent to infinite-depth feed-forward networks, and that they provide memory-efficient state-of-the-art performance for modeling sequential data. In [14], an alternative instantiation of an implicit layer is introduced by means of an ordinary differential equation (ODE). In this framework, termed "neural ODE," the output of the implicit layer is the solution to the underyling ODE. This is shown to be an expressive model class but requires solving an ordinary differential equation at test-time. Re-

cent works have extended various aspects of the implicit deep learning framework, e.g., to model graph-structured data [15], to multiscale modeling for image classification [16], and to estimate their Lipschitz constants for the purposes of robustness analysis [17]. While implicit models have been studied in a variety of contexts, their current training procedures rely upon end-to-end optimization approaches.

Conventional Deep Learning Deep CNNs have recently garnered a great deal of acclaim due to their success in the ImageNet competition [10], [18]–[20]. The emergence of deep architectures has moved the spotlight on a new set of challenges, including the vanishing gradient problem and parameter redundancy. In particular, architectures such as Residual Networks (ResNets) [18] and Highway networks [19] tackle the vanishing gradient problem by directly passing signals from one layer to the next using identity connections. This has been shown to improve both information and gradient flow in said deep networks. Moreover, techniques such as Stochastic Depth used in ResNets, where layers are randomly dropped during training, have also demonstrated that deep architectures are often plagued by parameter redundancy when having many layers [21]. In [20], the authors introduce the Dense Convolutional Network (DenseNet) for image classification, which extends the notion of skip connections used in ResNets and Highway networks such that all layers are connected directly with their subsequent layers. This network is composed of multiple dense block modules wherein all convolutional layers are linked in a forward-propagating manner. As a maximal amount of information is shared between layers in a dense block, DenseNets are known to have improved gradient propagation and parameter efficiency [20]. While [3] shows how CNNs and ResNets can be viewed as special cases of implicit models, we show that strictly upper triangular models can be viewed as a generalization of a dense block seen in DenseNets. This enables us to discuss the benefits of implicit models in terms of the benefits of DenseNets.

Layer-Wise Optimization The sequential, blockwise training approach proposed in this chapter can be seen as a generalization of layer-wise optimization for traditional neural networks. The paper [22] introduces greedy layer-wise training for deep neural networks, wherein the parameters are optimized per-layer in a sequential manner to pre-train an initialization of the network, which is then used in end-to-end optimization. This approach is motivated by the hypothesis that end-to-end gradient-based optimization is susceptible to becoming stuck in poor local solutions, whereas an individual layer's sub-optimization has a more benign landscape, which is justified by the authors experimentally. The work [23] extends the greedy layer-wise approach to convolutional neural networks, and finds that the learned model outperforms the state-of-the-art end-to-end training methods on the large-scale CIFAR-10 and ImageNet datasets.

Notations

Throughout this chapter, we define an implicit model using parameters $\mathbf{A} \in \mathbb{R}^{n\times n}$, $\mathbf{B} \in \mathbb{R}^{n\times p}$, $\mathbf{C} \in \mathbb{R}^{q\times n}$, and $\mathbf{D} \in \mathbb{R}^{q\times p}$ with nonlinear activation map $\phi\colon \mathbb{R}^n \to \mathbb{R}^n$. The element-wise absolute value of $\mathbf{A}$ is denoted by $|\mathbf{A}|$. The Perron-Frobenius eigenvalue of $|\mathbf{A}|$, i.e., the real eigenvalue

larger than the modulus of all other eigenvalues, is denoted by $\lambda_{\text{PF}}(|\mathbf{A}|)$. The input matrix with m input vectors $u \in \mathbb{R}^p$ is represented by $\mathbf{U} \in \mathbb{R}^{p\times m}$, and similarly the target matrix with m output vectors $y \in \mathbb{R}^q$ is represented by $\mathbf{Y} \in \mathbb{R}^{q\times m}$. The notation $\lfloor\alpha\rfloor$ represents the floor operator on scalar α. For a vector $x \in \mathbb{R}^n$ and natural number p, $\|x\|_p$ denotes the ℓ_p-norm of x, whereas for a matrix $\mathbf{M} \in \mathbb{R}^{m\times n}$, $\|\mathbf{M}\|_F$ denotes the Frobenius norm and $\|\mathbf{M}\|_p$ represents the ℓ_p-induced operator norm; $\|\mathbf{M}\|_p = \sup_{x\in\mathbb{R}^n\setminus\{0\}} \frac{\|\mathbf{M}x\|_p}{\|x\|_p}$. The rectified linear unit is $\mathrm{ReLU}(\cdot) = \max\{0, \cdot\}$, where, for the matrix $\mathbf{M}$, the maximum is taken element-wise. If $\mathbf{M} \in \mathbb{R}^{n\times n}$ is square, then we write $\mathrm{diag}(\mathbf{M})$ to mean the n-vector $(M_{11}, M_{22}, \ldots, M_{nn})$. Finally, $\mathcal{U}(a, b)$ denotes the uniform probability distribution with support $[a, b]$.

2.2 Background

Well-posedness of Implicit Models

The state x, characterized by the fixed-point equation, can be thought to represent the latent features extracted from the input [3]. In general, however, the fixed-point equation may not necessarily be well-posed in x—the activation map ϕ and matrix $\mathbf{A}$ must adhere to certain conditions to ensure the existence of a unique solution x to the fixed-point equation. Unique solutions to the fixed-point equation are desirable as this transfers to unique input-to-prediction mappings of the implicit model. The recent work [3] has introduced rigorous and numerically tractable conditions under which the fixed-point equation has a unique solution, which we recall in this section. In subsequent sections, we will show how such conditions can be incorporated into the training problem as constraints to guarantee the well-posedness of the learned model. We begin with a few definitions.

Definition 1. A matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ is said to be *well-posed with respect to* ϕ if, for all $b \in \mathbb{R}^n$, the equation

$$x = \phi(\mathbf{A}x + b) \tag{2.1}$$

has a unique solution $x \in \mathbb{R}^n$.

We write $\mathbf{A} \in \mathrm{WP}(\phi)$ to mean that $\mathbf{A}$ is well-posed with respect to ϕ.

Definition 2. A map $\phi\colon \mathbb{R}^n \to \mathbb{R}^n$ is called *component-wise non-expansive (CONE)* if $|\phi_i(x) - \phi_i(y)| \le |x_i - y_i|$ for all $x, y \in \mathbb{R}^n$ and all $i \in \{1, 2, \ldots, n\}$.

Note that ReLU, leaky ReLU, sigmoid, and $\tanh$ activation maps are all CONE maps. CONE maps allow for a simple sufficient condition to ensure that the fixed-point equation is well-posed:

Jibberish 1 (Well-Posedness of CONE Maps [3]). Assume that ϕ is a CONE map. If $\lambda_{\text{PF}}(|\mathbf{A}|) < 1$, then $\mathbf{A} \in \mathrm{WP}(\phi)$ and the fixed-point iteration $x(0) = 0,\ x(t+1) = \phi(\mathbf{A}x(t) + b),\ t \in \{0, 1, 2, \ldots\}$, converges to the unique solution of (2.1).

We remark that the set $\mathcal{A}_{\text{PF}} \coloneqq \{\mathbf{A} \in \mathbb{R}^{n\times n} : \lambda_{\text{PF}}(|\mathbf{A}|) < 1\}$ is not convex, but the inequality $\lambda_{\text{PF}}(|\mathbf{A}|) < \|\mathbf{A}\|_\infty$ implies that $\mathcal{A}_\infty \coloneqq \{\mathbf{A} \in \mathbb{R}^{n\times n} : \|\mathbf{A}\|_\infty < 1\}$ is a convex subset of $\mathcal{A}_{\text{PF}}$,

which is useful for efficiently treating the constraint $\mathbf{A} \in \mathrm{WP}(\phi)$ in the training problem. This work focuses on models where ϕ satisfies the CONE property.

Training Problem

We now turn our attention to formalizing the training problem for the implicit model. Suppose that we are given an input matrix $\mathbf{U} = \begin{bmatrix} u_1 & \cdots & u_m \end{bmatrix} \in \mathbb{R}^{p \times m}$ and a target matrix $\mathbf{Y} = \begin{bmatrix} y_1 & \cdots & y_m \end{bmatrix} \in \mathbb{R}^{q \times m}$. The prediction and fixed-point equation of the implicit model can then be written in matrix form as

$$\hat{\mathbf{Y}}(\mathbf{U}) = \mathbf{C}\mathbf{X} + \mathbf{D}\mathbf{U}, \tag{2.2}$$

$$\mathbf{X} = \phi(\mathbf{A}\mathbf{X} + \mathbf{B}\mathbf{U}), \tag{2.3}$$

where $\hat{\mathbf{Y}}(\mathbf{U}) = \begin{bmatrix} \hat{y}(u_1) & \cdots & \hat{y}(u_m) \end{bmatrix} \in \mathbb{R}^{q \times m}$. The training problem is then formulated as

$$\begin{array}{ll} \underset{\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D},\mathbf{X}}{\min} & \mathcal{L}(\mathbf{Y}, \mathbf{C}\mathbf{X} + \mathbf{D}\mathbf{U}) + \mathcal{P}(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}) \\ s_t & \mathbf{X} = \phi(\mathbf{A}\mathbf{X} + \mathbf{B}\mathbf{U}),\ \mathbf{A} \in \mathrm{WP}(\phi), \end{array} \tag{2.4}$$

where the loss function $\mathcal{L}$ is convex in its second argument, and $\mathcal{P}$ is an optional convex penalty function. As shown in Theorem 1, when ϕ is a CONE map, it suffices to replace the constraint $\mathbf{A} \in \mathrm{WP}(\phi)$ by the nonconvex constraint $\lambda_{\mathrm{PF}}(|\mathbf{A}|) < 1$. For efficient treatment, the nonconvex constraint can be relaxed to the convex constraint $\|\mathbf{A}\|_\infty < 1$.

2.3 Strictly Block Triangular Implicit Models

We begin by partitioning an implicit model of order n into $L > 1$ uniform parts as $n = n_1 + \cdots + n_L$, where $n_i = \lfloor \frac{n}{L} \rfloor$ for $i \in \{1, 2, \ldots, L-1\}$ and $n_L = n - \sum_{i=1}^{L-1} n_i$. We may write the model matrices in terms of blocks associated with each part of the partition:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{L,L} & \mathbf{A}_{L,L-1} & \cdots & \mathbf{A}_{L,1} \\ \mathbf{A}_{L-1,L} & \mathbf{A}_{L-1,L-1} & \cdots & \mathbf{A}_{L-1,1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{1,L} & \mathbf{A}_{1,L-1} & \cdots & \mathbf{A}_{1,1} \end{bmatrix}, \ \mathbf{B} = \begin{bmatrix} \mathbf{B}_L \\ \mathbf{B}_{L-1} \\ \vdots \\ \mathbf{B}_1 \end{bmatrix},$$

$$\mathbf{C} = \begin{bmatrix} \mathbf{C}_L & \mathbf{C}_{L-1} & \cdots & \mathbf{C}_1 \end{bmatrix}, \ \mathbf{X} = \begin{bmatrix} \mathbf{X}_L \\ \mathbf{X}_{L-1} \\ \vdots \\ \mathbf{X}_1 \end{bmatrix},$$

where $\mathbf{A}_{i,j} \in \mathbb{R}^{n_i \times n_j}$, $\mathbf{B}_i \in \mathbb{R}^{n_i \times p}$, $\mathbf{C}_i \in \mathbb{R}^{q \times n_i}$, and $\mathbf{X}_i \in \mathbb{R}^{n_i \times m}$ for all $i, j \in \{1, 2, \ldots, L\}$. Next, we impose the following assumptions on our model.

Assumption 1. The activation map ϕ is a CONE map.

As mentioned earlier, most common activation maps used in deep learning, e.g., ReLU, are CONE maps, and therefore Assumption 1 is not restrictive. Moreover, this assumption allows the use of convex constraints to enforce the desirable well-posedness guarantee upon the model.

Assumption 2. The prediction equation has no feed-through from the input; $\mathbf{D} = 0$.

Since feed-forward neural networks with non-polynomial activation maps, e.g., ReLU maps, are universal function approximators [24], and implicit models recover feed-forward neural networks with $\mathbf{D} = 0$ and appropriate choices of $\mathbf{A}, \mathbf{B}, \mathbf{C}$, it holds that the implicit model remains a universal function approximator under Assumption 2.

Assumption 3. The matrix $\mathbf{A}$ is strictly upper block triangular, i.e., $\mathbf{A}_{i,j} = 0$ for all pairs (i, j) with $i \leq j$.

Assumption 3 is not restrictive in practice, since an implicit model with such a matrix $\mathbf{A}$ is able to represent standard network architectures used today, e.g., feed-forward, convolutional, and residual networks [3]. In fact, feed-forward networks correspond to setting $\mathbf{A}_{i,j} = 0$ for all pairs (i, j) such that $j \neq i + 1$, and therefore it is easily shown that our strictly upper block triangular model matrix allows for precisely $(L-2)n_1n_L + \frac{1}{2}(L-2)(L-3)n_1^2$ more nonzero parameters than standard feed-forward models of the same model order n. Furthermore, Assumption 3 is justified from a computational standpoint, since the condition $\mathbf{A} \in \mathrm{WP}(\phi)$ is trivially satisfied as ϕ is a CONE map, allowing us to remove this constraint from the training problem.

Relation to DenseNets

While [3] has already established how implicit models can encapsulate a significant subset of conventional architectures, we now highlight the relationship between implicit models with strictly upper block triangular structure and dense block modules that constitute DenseNets as conceived in [20].

Figure 2.1 illustrates a computational graph detailing how input information is propagated through an implicit model under Assumptions 1–3. Note that this computational graph is characterized by a forward-propagating structure with an exhaustive set of $L(L + 1)/2$ generalized, weighted skip connections linking each pair of feature blocks including the input. Differences to dense blocks, as proposed in [20], include that implicit models with said structure allow for arbitrary linear transformations (not only convolutions) between feature blocks as well as weighted skip connections throughout the model, including from the input to every feature block. This shows that strictly upper block triangular implicit models are a generalization of dense blocks.

As a result of this characterization, implicit models with an upper block triangular structure directly inherit many of the benefits observed in DenseNets. Due to the exhaustive set of connections between each feature block, the considered implicit models have maximal information flow. As a consequence, such models are also bound to have a greater parameter efficiency. Finally, since we connect each feature block directly to the output, the training procedure for such models will benefit from improved gradient propagation.

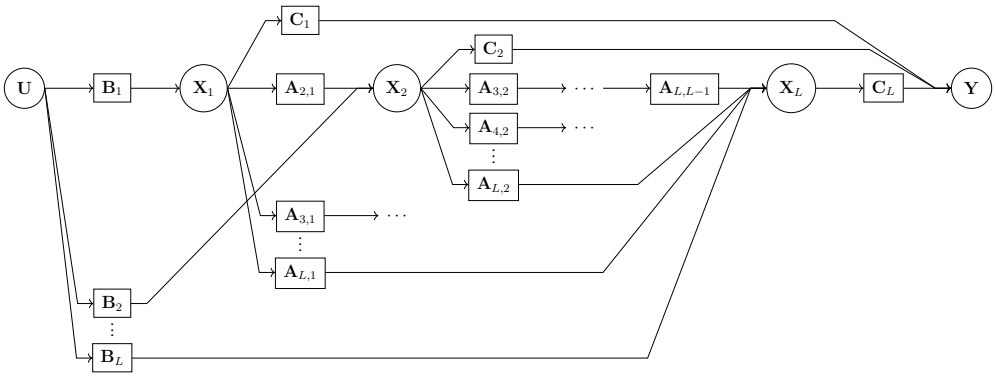


Figure 2.1: Computational graph for an implicit model with a strictly upper block triangular $\mathbf{A}$ matrix.

2.4 Sequential Blockwise Training

Our approach is a greedy, sequential blockwise training method for strictly upper block triangular implicit models. The training problem under Assumptions 1-3 reduces to

$$\begin{array}{ll} \min\limits_{\substack{\{\mathbf{A}_{i,j}\}_{1\le j<i\le L},\\ \{\mathbf{B}_i,\mathbf{C}_i,\mathbf{X}_i\}_{i=1}^{L}}} & \mathcal{L}(\mathbf{Y}, \sum_{j=1}^{L} \mathbf{C}_j\mathbf{X}_j) \\ s_t & \mathbf{X}_1 = \phi(\mathbf{B}_1\mathbf{U}), \\ & \mathbf{X}_i = \phi(\sum_{j=1}^{i-1} \mathbf{A}_{i,j}\mathbf{X}_j + \mathbf{B}_i\mathbf{U}), \\ & i \in \{2,\dots,L\}. \end{array} \tag{2.5}$$

With the partition of n into L blocks, we may iteratively increment the model order of the implicit model by n_i and train each added block individually, while holding previously optimized fixed-point parameters constant and re-optimizing the auxiliary parameters. We formalize this approach in Algorithm 1. Due to the strictly upper block triangular structure assumed upon $\mathbf{A}$, the first subproblem (2.6) only optimizes over subblocks of the $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{X}$ matrices. The i^{th} subproblem optimizes over $\{\mathbf{A}_{i,j}\}_{j=1}^{i-1}$, $\mathbf{B}_i$, $\mathbf{X}_i$ and auxiliary blocks $\{\mathbf{C}_j\}_{j=1}^{i}$. Here feature blocks $\{\mathbf{X}_j\}_{j=1}^{i-1}$ are fixed and obtained as optimizers of the previous subproblems. The sequence of optimizations in Algorithm 1 occurs only once, so that the obtained subblocks can be concatenated to form the model matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$. Figure 2.2 depicts a visual representation of the optimization sequence.

Remark 1. Our proposed approach can be applied to both regression and classification tasks. For regression, the convex loss can be chosen to be the mean-squared error (MSE) loss

$$\mathcal{L}(\mathbf{Y}, \hat{\mathbf{Y}}) = \frac{1}{2}\|\mathbf{Y} - \hat{\mathbf{Y}}\|_F^2, \tag{2.8}$$

Algorithm 1 Sequential Greedy Training for Implicit Models

Input: Input $\mathbf{U} \in \mathbb{R}^{p\times m}$, target $\mathbf{Y} \in \mathbb{R}^{q\times m}$.
Parameters: Model order $n > 1$, partition size $L > 1$.
Design choices: Loss $\mathcal{L}$, activation map ϕ.
Return: $\mathbf{A} \in \mathbb{R}^{n\times n}$, $\mathbf{B} \in \mathbb{R}^{n\times p}$, $\mathbf{C} \in \mathbb{R}^{q\times n}$.

begin training
1. Partition $n = n_1 + \cdots + n_L$ with $n_i = \lfloor \frac{n}{L} \rfloor$ for $i \in \{1, 2, \ldots, L-1\}$ and $n_L = n - \sum_{i=1}^{L-1} n_i$.
2. Solve optimization

$$\min_{\mathbf{B}_1, \mathbf{C}_1, \mathbf{X}_1} \mathcal{L}(\mathbf{Y}, \mathbf{C}_1\mathbf{X}_1) \; s_t \; \mathbf{X}_1 = \phi(\mathbf{B}_1\mathbf{U}). \tag{2.6}$$

for $i \in \{2, 3, \ldots, L\}$ **do**
3. Solve optimization

$$\begin{array}{cc} \min\limits_{\substack{\{\mathbf{A}_{i,j}\}_{j=1}^{i-1}, \mathbf{B}_i, \\ \{\mathbf{C}_j\}_{j=1}^{i}, \mathbf{X}_i}} & \mathcal{L}(\mathbf{Y}, \sum_{j=1}^{i} \mathbf{C}_j\mathbf{X}_j) \\ s_t & \mathbf{X}_i = \phi(\sum_{j=1}^{i-1} \mathbf{A}_{i,j}\mathbf{X}_j + \mathbf{B}_i\mathbf{U}). \end{array} \tag{2.7}$$

end for
end training

while multi-class classification can be accommodated using a combination of the softmax and negative cross-entropy loss:

$$\mathcal{L}(\mathbf{Y}, \hat{\mathbf{Y}}) = \log(\mathbf{1}^\top \times \exp(\hat{\mathbf{Y}}))\mathbf{1} - \mathrm{Tr}(\mathbf{Y}^\top\hat{\mathbf{Y}}). \tag{2.9}$$

Remark 2. Each subproblem can be solved using gradient-based local search heuristics implemented with automatic differentiation software. Note that the i^{th} subproblem can be decomposed into a regression upon the target using both a feedforward component and direct linear (skip) contributions from the previous feature blocks. As such, this sequential approach avoids the need for implicit differentiation.

Remark 3. Even though our approach sets up a uniform partition of the model order n, this partition can be chosen arbitrarily. The reason the algorithm is initialized with a uniform partition is that the training complexity is equally distributed amongst all subproblems.

Remark 4. Inherent to our approach is the restriction of $\mathbf{A}$ as a strictly upper block triangular matrix. This assumption alleviates the requirement to project $\mathbf{A}$ onto the well-posedness ball at each iteration of the gradient-based optimization method. In particular, when considering the convex constraint $\|\mathbf{A}\|_\infty < 1$ sufficient for well-posedness, our method saves running a bisection method with complexity $O(n^3)$ at each projected gradient update.

$$\begin{bmatrix} \mathbf{X}_L \\ \hline \mathbf{X}_{L-1} \\ \hline \vdots \\ \hline \mathbf{X}_2 \\ \hline \mathbf{X}_1 \end{bmatrix} = \phi \left(\left[\begin{array}{c|c|c|c|c} 0 & \mathbf{A}_{L,L-1} & \cdots & \mathbf{A}_{L,2} & \mathbf{A}_{L,1} \\ \hline 0 & 0 & \cdots & \mathbf{A}_{L-1,2} & \mathbf{A}_{L-1,1} \\ \hline \vdots & \vdots & \ddots & \vdots & \vdots \\ \hline 0 & 0 & \cdots & 0 & \mathbf{A}_{2,1} \\ \hline 0 & 0 & \cdots & 0 & 0 \end{array}\right] \begin{bmatrix} \mathbf{X}_L \\ \hline \mathbf{X}_{L-1} \\ \hline \vdots \\ \hline \mathbf{X}_2 \\ \hline \mathbf{X}_1 \end{bmatrix} + \begin{bmatrix} \mathbf{B}_L \\ \hline \mathbf{B}_{L-1} \\ \hline \vdots \\ \hline \mathbf{B}_2 \\ \hline \mathbf{B}_1 \end{bmatrix} \mathbf{U} \right)$$

$$\mathbf{Y} = \left[\begin{array}{c|c|c|c|c} \mathbf{C}_L & \mathbf{C}_{L-1} & \cdots & \mathbf{C}_2 & \mathbf{C}_1 \end{array}\right] \begin{bmatrix} \mathbf{X}_L \\ \hline \mathbf{X}_{L-1} \\ \hline \vdots \\ \hline \mathbf{X}_2 \\ \hline \mathbf{X}_1 \end{bmatrix}$$

Figure 2.2: Illustration of the proposed sequential blockwise training scheme. Blocks in (light) green are optimized initially, whereas the blocks in (dark) blue are optimized towards the end of the algorithm. The auxiliary $\mathbf{C}$ blocks are re-optimized at every iteration.

Remark 5. Our approach puts less strain on the memory requirements during the training procedure as compared to the end-to-end optimization. For the former, the i^{th} subproblem only requires optimizing a subset of $n_i(\sum_{j=1}^{i-1} n_j + m + p) + q\sum_{j=1}^{i} n_j$ parameters, whereas the latter optimizes over $n(n+m+p+q)$ parameters simultaneously. In the case where we have a large order model and $n \gg n_i$, each optimization of the blockwise method needs to update significantly fewer parameters at a time than in the end-to-end method.

Remark 6. The greedy blockwise approach also lends itself to an increased interpretability of the implicit model parameters. Within an end-to-end training approach of a dense implicit model, we lack insight into how the parameters are working together to minimize the loss. Our approach directly supervises the training for each subblock of parameters. In the first optimization, blocks $\mathbf{B}_1$, $\mathbf{C}_1$, and $\mathbf{X}_1$ attempt to directly regress upon the target. For the i^{th} optimization, the additional fixed-point parameters $\{\mathbf{A}_{i,j}\}_{j=1}^{i-1}$, $\mathbf{B}_i$, and $\mathbf{X}_i$ attempt to improve the regression by offering greater modeling capacity while auxiliary parameters $\{\mathbf{C}_j\}_{j=1}^{i}$ re-weight the contribution of each feature block $\{\mathbf{X}_j\}_{j=1}^{i}$.

Alternating Minimization for Implicit Models

While local search presents a valid means to solve the subproblems of Algorithm 1, we also propose an efficient alternating minimization heuristic for models trained under the squared Euclidean loss.

Assumption 4. The training problem uses the squared Euclidean loss (2.8).

When considering the training problem of implicit models with Euclidean loss, we can formu-

late the i^{th} subproblem in Algorithm 1 as follows:

$$\begin{array}{ll} \underset{\substack{\{\mathbf{A}_{i,j}\}_{j=1}^{i-1},\mathbf{B}_i,\\ \{\mathbf{C}_j\}_{j=1}^{i},\mathbf{X}_i}}{\min} & \|\mathbf{Y}-\sum_{j=1}^{i}\mathbf{C}_j\mathbf{X}_j\|_F^2 \\ s_t & \mathbf{X}_i=\phi(\sum_{j=1}^{i-1}\mathbf{A}_{i,j}\mathbf{X}_j+\mathbf{B}_i\mathbf{U}). \end{array} \tag{2.10}$$

Rather than using a local search method to solve (2.10), we can leverage an alternating minimization heuristic. This subroutine is formalized in Algorithm 2. It decomposes each subproblem into alternating between solving a least squares problem and a single hidden-layer neural network training problem, each with global optimality guarantees. Each least squares problem (3.14) can be solved to global optimality with a computational complexity of $O\left((\sum_{j=1}^{i} n_j)^3\right)$, while there exist guarantees for solving the shallow training problem to (near) global optimality via local search heuristics [25], [26].

Extension to Non-Strict Upper Triangular ReLU Models

In this section, we relax Assumption 3 to allow for $\mathbf{A}$ to be upper triangular, i.e., we now assume that $\mathbf{A}_{i,j}=0$ for all pairs (i,j) with $i<j$ and $\mathbf{A}_{i,i}$ is upper triangular for all i. This new model structure allows for self-loops at all of the feature blocks $\mathbf{X}_i$ (i.e., self-loops at the circular nodes in Figure 2.1). In this case, the i^{th} subproblem (2.7) in the blockwise sequential approach becomes

$$\begin{array}{ll} \underset{\substack{\{\mathbf{A}_{i,j}\}_{j=1}^{i},\mathbf{B}_i,\\ \{\mathbf{C}_j\}_{j=1}^{i},\mathbf{X}_i}}{\min} & \mathcal{L}(\mathbf{Y},\sum_{j=1}^{i}\mathbf{C}_j\mathbf{X}_j) \\ s_t & \mathbf{X}_i=\phi(\sum_{j=1}^{i}\mathbf{A}_{i,j}\mathbf{X}_j+\mathbf{B}_i\mathbf{U}), \\ & -1<\mathrm{diag}(\mathbf{A}_{i,i})<1, \\ & (\mathbf{A}_{i,i})_{kl}=0 \text{ for all } k>l, \end{array} \tag{2.13}$$

for $i\in\{1,2,\ldots,L\}$, where the constraints $-1<\mathrm{diag}(\mathbf{A}_{i,i})<1$ are linear constraints ensuring that the triangular matrix $\mathbf{A}$ is well-posed with respect to ϕ, as guaranteed by Theorem 1. The difficulty in the new problem (2.13) is the fact that the constraint on the variable $\mathbf{X}_i$ is now an implicit equality constraint. In general, this requires the use of implicit differentiation at each step of a gradient-based optimization algorithm, making each of the subproblems nontrivial to solve, and eliminating the immediate applicability of the alternating subroutine of Algorithm 2. Despite this challenge, we prove in Theorem 2 that for ReLU models, the feasible set of (2.13) is equivalently expressed in terms of an explicit equality constraint, making the i^{th} subproblem equivalent to a re-weighted subproblem under our original strictly upper block triangular assumption. For the sake of exposition, we assume that $n_j=1$. We generalize the result to arbitrary n_j in Section 2.6.

Jibberish 2. Assume that $\phi=\mathrm{ReLU}$ and $n_j=1$ for all $j\in\{1,2,\ldots,L\}$, and let $\mathbf{X}_i\in\mathbb{R}^{1\times m}$. Then, there exist $\{\mathbf{A}_{i,j}\}_{j=1}^{i},\mathbf{B}_i,\{\mathbf{C}_j\}_{j=1}^{i}$ that together with $\mathbf{X}_i$ are feasible for (2.13) if and only if there exist $\gamma_1,\ldots,\gamma_{i-1},\lambda_1,\ldots,\lambda_p\in\mathbb{R}$ such that $\mathbf{X}_i=\mathrm{ReLU}\left(\sum_{j=1}^{i-1}\gamma_j\mathbf{X}_j+\sum_{k=1}^{p}\lambda_k\mathbf{U}_k\right)$, where $\mathbf{U}_k$ is the k^{th} row of $\mathbf{U}$.

Algorithm 2 Alternating Minimization Subroutine

Input: Features $\{\mathbf{X}_j\}_{j=1}^{i-1}$ from first $i-1$ subproblems.
Parameters: Number of alternating iterations T.
Return: $\{\mathbf{A}_{i,j}\}_{j=1}^{i-1}$, $\mathbf{B}_i$, $\mathbf{X}_i$, and $\{\mathbf{C}_j\}_{j=1}^{i}$.

begin subroutine
1. Initialize $\mathbf{X}_i$.
for $t \in \{1, 2, \ldots, T\}$ **do**
2. Solve least squares optimization

$$\min_{\{\mathbf{C}_j\}_{j=1}^{i}} \|\mathbf{Y} - \sum_{j=1}^{i} \mathbf{C}_j \mathbf{X}_j\|_F^2. \tag{2.11}$$

3. Solve single hidden-layer ReLU training problem

$$\begin{aligned} \min_{\tilde{\mathbf{A}}_i, \mathbf{X}_i} \ & \|\tilde{\mathbf{Y}} - \mathbf{C}_i \mathbf{X}_i\|_F^2 \\ s_t \ & \mathbf{X}_i = \phi(\tilde{\mathbf{A}}_i \tilde{\mathbf{U}}_i), \end{aligned} \tag{2.12}$$

with $\tilde{\mathbf{Y}} = \mathbf{Y} - \sum_{j=1}^{i-1} \mathbf{C}_j \mathbf{X}_j$,

$$\tilde{\mathbf{A}}_i = \begin{bmatrix} \mathbf{A}_{i,i-1} & \cdots & \mathbf{A}_{i,1} & \mathbf{B}_i \end{bmatrix},$$

and

$$\tilde{\mathbf{U}}_i = \begin{bmatrix} \mathbf{X}_{i-1}^\top & \cdots & \mathbf{X}_1^\top & \mathbf{U}^\top \end{bmatrix}^\top.$$

end for
end subroutine

Proof. Notice that, since $n_j = 1$ for all j, every block of $\mathbf{A}$ is a scalar, so we write $\mathbf{A}_{i,j} = a_{ij}$. Similarly, denote the kth element of the row vector $\mathbf{B}_i$ by b_{ik}.

Suppose that $\{a_{ij}\}_{j=1}^{i}$, $\mathbf{B}_i$, and $\{\mathbf{C}_j\}_{j=1}^{i}$, together with $\mathbf{X}_i$, are feasible for (2.13). Then $-1 < a_{ii} < 1$ and

$$\mathbf{X}_i = \mathrm{ReLU}\left(\sum_{j=1}^{i} a_{ij} \mathbf{X}_j + \sum_{k=1}^{p} b_{ik} \mathbf{U}_k \right).$$

Let $l \in \{1, 2, \ldots, m\}$. If $X_{il} \neq 0$, then we have that

$$\begin{aligned} 0 < X_{il} &= \mathrm{ReLU}\left(\sum_{j=1}^{i} a_{ij} X_{jl} + \sum_{k=1}^{p} b_{ik} U_{kl}\right) \\ &= \sum_{j=1}^{i} a_{ij} X_{jl} + \sum_{k=1}^{p} b_{ik} U_{kl}. \end{aligned}$$

Hence, $X_{il} = \sum_{j=1}^{i-1} \frac{a_{ij}}{1-a_{ii}} X_{jl} + \sum_{k=1}^{p} \frac{1}{1-a_{ii}} b_{ik} U_{kl} = \mathrm{ReLU}\left(\sum_{j=1}^{i-1} \frac{a_{ij}}{1-a_{ii}} X_{jl} + \sum_{k=1}^{p} \frac{1}{1-a_{ii}} b_{ik} U_{kl}\right)$. On the other hand, if $X_{il} = 0$, then $\sum_{j=1}^{i} a_{ij} X_{jl} + \sum_{k=1}^{p} b_{ik} U_{kl} = \sum_{j=1}^{i-1} a_{ij} X_{jl} + \sum_{k=1}^{p} b_{ik} U_{kl} \leq 0$, so $\sum_{j=1}^{i-1} \frac{a_{ij}}{1-a_{ii}} X_{jl} + \sum_{k=1}^{p} \frac{b_{ik}}{1-a_{ii}} U_{kl} \leq 0$, which implies that again

$$X_{il} = \mathrm{ReLU}\left(\sum_{j=1}^{i-1} \frac{a_{ij}}{1-a_{ii}} X_{jl} + \sum_{k=1}^{p} \frac{b_{ik}}{1-a_{ii}} U_{kl}\right).$$

Thus, it holds that

$$\mathbf{X}_i = \mathrm{ReLU}\left(\sum_{j=1}^{i-1} \frac{a_{ij}}{1-a_{ii}} \mathbf{X}_j + \sum_{k=1}^{p} \frac{b_{ik}}{1-a_{ii}} \mathbf{U}_k\right),$$

which proves the forward direction with $\gamma_j = \frac{a_{ij}}{1-a_{ii}}$ and $\lambda_k = \frac{b_{ik}}{1-a_{ii}}$.

Now, suppose there exist $\gamma_1, \ldots, \gamma_{i-1}, \lambda_1, \ldots, \lambda_p \in \mathbb{R}$ such that

$$\mathbf{X}_i = \mathrm{ReLU}\left(\sum_{j=1}^{i-1} \gamma_j \mathbf{X}_j + \sum_{k=1}^{p} \lambda_k \mathbf{U}_k\right).$$

Then, let $\epsilon > 0$ and define $\{\mathbf{A}_{i,j}\}_{j=1}^{i} \subseteq \mathbb{R}$, $\mathbf{B}_i \in \mathbb{R}^{1\times p}$ by $\mathbf{A}_{i,i} = a_{ii} := 1-\epsilon$, $\mathbf{A}_{i,j} = (1-a_{ii})\gamma_j$ for all $j \in \{1, 2, \ldots, i-1\}$, and $(\mathbf{B}_i)_k = (1-a_{ii})\lambda_k$ for all $k \in \{1, 2, \ldots, p\}$. Let $\{\mathbf{C}_j\}_{j=1}^{i} \subseteq \mathbb{R}^{q\times 1}$ be arbitrary. Then, by construction $-1 < \mathrm{diag}(\mathbf{A}_{i,i}) = a_{ii} = 1-\epsilon < 1$ and $(\mathbf{A}_{i,i})_{kl} = 0$ for all $k > l$. Reversing the steps of the forward direction proof also shows that our parameter choice yields $\mathbf{X}_i = \mathrm{ReLU}\left(\sum_{j=1}^{i} \mathbf{A}_{i,j}\mathbf{X}_j + \mathbf{B}_i\mathbf{U}\right)$. Thus, $\{\mathbf{A}_{i,j}\}_{j=1}^{i}, \mathbf{B}_i, \{\mathbf{C}_j\}_{j=1}^{i}$, together with $\mathbf{X}_i$, are feasible for (2.13). □

Theorem 2 shows that the implicit constraint in the non-strict training problem (2.13) may be equivalently replaced by an explicit constraint, avoiding the need to use implicit differentiation in solving the optimization for non-strict models.

2.5 Conclusion

In this chapter, we introduce a sequential, greedy algorithm for training triangular implicit deep models in a blockwise fashion. We show how the corresponding subproblems can be decomposed

into a subroutine that alternates between a least squares optimization and an easily solved single hidden-layer neural network training problem. We theoretically prove that, for the more general non-strictly triangular ReLU implicit models, the challenging implicit constraints can be equivalently replaced by more tractable explicit constraints, allowing for our algorithm to be applied to such models. Experiments on function interpolation, as well as MNIST and Fashion-MNIST classification tasks, show that our algorithm learns models with superior performance, parameter efficiency and training time as compared to end-to-end optimization of dense implicit models. This makes the proposed sequential algorithm a promising new approach for training implicit deep models.

2.6 Supplementary Information: Generalization of Theorem 2

Below, we generalize Theorem 2 to the case where the partition sizes n_j may be larger than unity. The result shows that, again, the implicit constraint on $\mathbf{X}_i$ may be replaced by equivalent explicit ones.

Jibberish 3. Assume that $\phi = \mathrm{ReLU}$, and let $\mathbf{X}_i \in \mathbb{R}^{n_i \times m}$. Then, there exists $\{\mathbf{A}_{i,j}\}_{j=1}^{i}, \mathbf{B}_i, \{\mathbf{C}_j\}_{j=1}^{i}$ that together with $\mathbf{X}_i$ are feasible for (2.13) if and only if, for all $k \in \{1, 2, \ldots, n_i\}$ and all $l \in \{1, 2, \ldots, m\}$, there exists $\gamma_{k1}^{(1)}, \ldots, \gamma_{kn_1}^{(1)} \in \mathbb{R}$, …, $\gamma_{k1}^{(i-1)}, \ldots, \gamma_{kn_{i-1}}^{(i-1)} \in \mathbb{R}$, $\gamma_{k(k+1)}^{(i)}, \ldots, \gamma_{kn_i}^{(i)} \in \mathbb{R}$, and $\lambda_{k1}, \ldots, \lambda_{kp} \in \mathbb{R}$ such that

$$\begin{aligned}(\mathbf{X}_i)_{kl} = \mathrm{ReLU}\Bigg(&\sum_{r=k+1}^{n_i} \gamma_{kr}^{(i)} (\mathbf{X}_i)_{rl} \\ &+ \sum_{j=1}^{i-1} \sum_{r=1}^{n_j} \gamma_{kr}^{(j)} (\mathbf{X}_j)_{rl} + \sum_{r=1}^{p} \lambda_{kr} (\mathbf{U})_{rl} \Bigg).\end{aligned} \tag{2.14}$$

Proof. Suppose that $\{\mathbf{A}_{i,j}\}_{j=1}^{i}, \mathbf{B}_i, \{\mathbf{C}_j\}_{j=1}^{i}$, together with $\mathbf{X}_i$, are feasible for (2.13). Then $-1 < \mathrm{diag}(\mathbf{A}_{i,i}) < 1$, $(\mathbf{A}_{i,i})_{kl} = 0$ for all $k > l$, and

$$\mathbf{X}_i = \mathrm{ReLU}\left(\sum_{j=1}^{i} \mathbf{A}_{i,j} \mathbf{X}_j + \mathbf{B}_i \mathbf{U} \right).$$

Let $k \in \{1, 2, \ldots, n_i\}$ and $l \in \{1, 2, \ldots, m\}$. If $(\mathbf{X}_i)_{kl} \neq 0$, then we have that

$$\begin{aligned}
0 &< (\mathbf{X}_i)_{kl} \\
&= \mathrm{ReLU}\left(\sum_{j=1}^{i} \sum_{r=1}^{n_j} (\mathbf{A}_{i,j})_{kr} (\mathbf{X}_j)_{rl} + \sum_{r=1}^{p} (\mathbf{B}_i)_{kr} (\mathbf{U})_{rl} \right) \\
&= \sum_{j=1}^{i} \sum_{r=1}^{n_j} (\mathbf{A}_{i,j})_{kr} (\mathbf{X}_j)_{rl} + \sum_{r=1}^{p} (\mathbf{B}_i)_{kr} (\mathbf{U})_{rl} \\
&= \sum_{r=1}^{n_i} (\mathbf{A}_{i,i})_{kr} (\mathbf{X}_i)_{rl} + \sum_{j=1}^{i-1} \sum_{r=1}^{n_j} (\mathbf{A}_{i,j})_{kr} (\mathbf{X}_j)_{rl} \\
&\qquad + \sum_{r=1}^{p} (\mathbf{B}_i)_{kr} (\mathbf{U})_{rl} \\
&= \sum_{r=k}^{n_i} (\mathbf{A}_{i,i})_{kr} (\mathbf{X}_i)_{rl} + \sum_{j=1}^{i-1} \sum_{r=1}^{n_j} (\mathbf{A}_{i,j})_{kr} (\mathbf{X}_j)_{rl} \\
&\qquad + \sum_{r=1}^{p} (\mathbf{B}_i)_{kr} (\mathbf{U})_{rl},
\end{aligned}$$

so

$$\begin{aligned}
(\mathbf{X}_i)_{kl} &= \sum_{r=k+1}^{n_i} \frac{(\mathbf{A}_{i,i})_{kr}}{1 - (\mathbf{A}_{i,i})_{kk}} (\mathbf{X}_i)_{rl} \\
&\qquad + \sum_{j=1}^{i-1} \sum_{r=1}^{n_j} \frac{(\mathbf{A}_{i,j})_{kr}}{1 - (\mathbf{A}_{i,i})_{kk}} (\mathbf{X}_j)_{rl} \\
&\qquad + \sum_{r=1}^{p} \frac{(\mathbf{B}_i)_{kr}}{1 - (\mathbf{A}_{i,i})_{kk}} (\mathbf{U})_{rl} \\
&= \mathrm{ReLU}\left(\sum_{r=k+1}^{n_i} \frac{(\mathbf{A}_{i,i})_{kr}}{1 - (\mathbf{A}_{i,i})_{kk}} (\mathbf{X}_i)_{rl} \right. \\
&\qquad + \sum_{j=1}^{i-1} \sum_{r=1}^{n_j} \frac{(\mathbf{A}_{i,j})_{kr}}{1 - (\mathbf{A}_{i,i})_{kk}} (\mathbf{X}_j)_{rl} \\
&\qquad \left. + \sum_{r=1}^{p} \frac{(\mathbf{B}_i)_{kr}}{1 - (\mathbf{A}_{i,i})_{kk}} (\mathbf{U})_{rl} \right).
\end{aligned}$$

On the other hand, if $(\mathbf{X}_i)_{kl} = 0$, then

$$\begin{aligned}
0 &\geq \sum_{j=1}^{i}\sum_{r=1}^{n_j}(\mathbf{A}_{i,j})_{kr}(\mathbf{X}_j)_{rl} + \sum_{r=1}^{p}(\mathbf{B}_i)_{kr}(\mathbf{U})_{rl} \\
&= \sum_{r=1}^{n_i}(\mathbf{A}_{i,i})_{kr}(\mathbf{X}_i)_{rl} + \sum_{j=1}^{i-1}\sum_{r=1}^{n_j}(\mathbf{A}_{i,j})_{kr}(\mathbf{X}_j)_{rl} \\
&\qquad + \sum_{r=1}^{p}(\mathbf{B}_i)_{kr}(\mathbf{U})_{rl} \\
&= \sum_{r=k+1}^{n_i}(\mathbf{A}_{i,i})_{kr}(\mathbf{X}_i)_{rl} + \sum_{j=1}^{i-1}\sum_{r=1}^{n_j}(\mathbf{A}_{i,j})_{kr}(\mathbf{X}_j)_{rl} \\
&\qquad + \sum_{r=1}^{p}(\mathbf{B}_i)_{kr}(\mathbf{U})_{rl},
\end{aligned}$$

so

$$\begin{aligned}
0 \geq \sum_{r=k+1}^{n_i}\frac{(\mathbf{A}_{i,i})_{kr}}{1-(\mathbf{A}_{i,i})_{kk}}(\mathbf{X}_i)_{rl} + \sum_{j=1}^{i-1}\sum_{r=1}^{n_j}\frac{(\mathbf{A}_{i,j})_{kr}}{1-(\mathbf{A}_{i,i})_{kk}}(\mathbf{X}_j)_{rl} \\
+ \sum_{r=1}^{p}\frac{(\mathbf{B}_i)_{kr}}{1-(\mathbf{A}_{i,i})_{kk}}(\mathbf{U})_{rl},
\end{aligned}$$

which implies that again

$$\begin{aligned}
(\mathbf{X}_i)_{kl} = \mathrm{ReLU}\Bigg(\sum_{r=k+1}^{n_i}\frac{(\mathbf{A}_{i,i})_{kr}}{1-(\mathbf{A}_{i,i})_{kk}}(\mathbf{X}_i)_{rl} \\
+ \sum_{j=1}^{i-1}\sum_{r=1}^{n_j}\frac{(\mathbf{A}_{i,j})_{kr}}{1-(\mathbf{A}_{i,i})_{kk}}(\mathbf{X}_j)_{rl} \\
+ \sum_{r=1}^{p}\frac{(\mathbf{B}_i)_{kr}}{1-(\mathbf{A}_{i,i})_{kk}}(\mathbf{U})_{rl}\Bigg).
\end{aligned}$$

Thus, (2.14) holds with $\gamma_{kr}^{(j)} = \frac{(\mathbf{A}_{i,j})_{kr}}{1-(\mathbf{A}_{i,i})_{kk}}$ and $\lambda_{kr} = \frac{(\mathbf{B}_i)_{kr}}{1-(\mathbf{A}_{i,i})_{kk}}$, which proves the forward direction.

Now, suppose that, for all $k \in \{1, 2, \ldots, n_i\}$ and all $l \in \{1, 2, \ldots, m\}$, there exists $\gamma_{k1}^{(1)}, \ldots, \gamma_{kn_1}^{(1)} \in \mathbb{R}$, $\ldots$, $\gamma_{k1}^{(i-1)}, \ldots, \gamma_{kn_{i-1}}^{(i-1)} \in \mathbb{R}$, $\gamma_{k(k+1)}^{(i)}, \ldots, \gamma_{kn_i}^{(i)} \in \mathbb{R}$, and $\lambda_{k1}, \ldots, \lambda_{kp} \in \mathbb{R}$ such that (2.14) holds. Then, let $\epsilon > 0$ and define $\mathbf{A}_{i,j} \in \mathbb{R}^{n_i \times n_j}$ for $j \in \{1, 2, \ldots, i\}$ and $\mathbf{B}_i \in \mathbb{R}^{n_i \times p}$ by $(\mathbf{A}_{i,i})_{kk} = 1 - \epsilon$ for all $k \in \{1, 2, \ldots, n_i\}$, $(\mathbf{A}_{i,i})_{kl} = 0$ for all $k > l$, $(\mathbf{A}_{i,i})_{kr} = (1 - (\mathbf{A}_{i,i})_{kk})\gamma_{kr}^{(i)}$ for all $k < r$, $(\mathbf{A}_{i,j})_{kr} = (1 - (\mathbf{A}_{i,i})_{kk})\gamma_{kr}^{(j)}$ for all $k \in \{1, 2, \ldots, n_i\}$, all $r \in \{1, 2, \ldots, n_j\}$,

and all $j \in \{1, 2, \ldots, i-1\}$, and $(\mathbf{B}_i) = (1 - (\mathbf{A}_{i,i})_{kk})\lambda_{kr}$ for all $k \in \{1, 2, \ldots, n_i\}$ and all $r \in \{1, 2, \ldots, p\}$. Let $\{\mathbf{C}_j\}_{j=1}^{i} \subseteq \mathbb{R}^{q \times 1}$ be arbitrary. Then, it is clear by construction that $-1 < \operatorname{diag}(\mathbf{A}_{i,i}) = (1-\epsilon)\mathbf{1}_{n_i} < 1$ (where $\mathbf{1}_{n_i}$ is the n_i-vector of all ones), and $(\mathbf{A}_{i,i})_{kl} = 0$ for all $k > l$. Reversing the steps of the forward direction proof also shows that our choice of parameters yields $\mathbf{X}_i = \operatorname{ReLU}\left(\sum_{j=1}^{i} \mathbf{A}_{i,j}\mathbf{X}_j + \mathbf{B}_i\mathbf{U}\right)$, so $\{\mathbf{A}_{i,j}\}_{j=1}^{i}, \mathbf{B}_i, \{\mathbf{C}_j\}_{j=1}^{i}$, together with $\mathbf{X}_i$, are feasible for (2.13). □

Chapter 3

Zeroth-Order Methods for Fine-tuning Language Models

3.1 Introduction

In recent years, language models (LMs) have exhibited exceptional performance in a vast array of domains within natural language processing (NLP) [27]–[29].This development has generated immense excitement within the research community and has propelled the advancement of the aforementioned models to the forefront of deep learning research.

Fine-tuning LMs has been the dominant strategy for adapting pre-trained models to specialized downstream tasks [30]. Fine-tuning often relies on first-order methods, such as stochastic gradient descent (SGD) [31] or Adam [32]. However, as LMs are scaled up, backpropagation [33] becomes prohibitive in terms of memory requirements. More concretely, [34] show that fine-tuning an OPT-13B model with full-parameter or parameter efficient fine-tuning (PEFT) using Adam requires $12\times$ and $6\times$ more memory than inference, respectively. This is due to the need to cache activations during the forward pass as well as gradients and optimizer states during the backward pass. This has given rise to memory-efficient inference-based adaptation methods, including in-context learning (ICL) and zeroth-order (ZO) optimization.

While ZO methods have been studied for decades [35], [36], it is only recently that these have been applied to fine-tune LMs [34]. In [34], authors propose the Memory-Efficient Zeroth-Order Optimizer (MeZO) and demonstrate its superior performance against ICL with a memory footprint equivalent to that of inference. By virtue of estimating gradients through loss computations, ZO methods are compatible with settings where gradients are non-accessible or infeasible to compute, e.g. when considering non-differentiable objectives or black-box access of LMs.

However, ZO methods still face challenges in large-scale settings. According to [34], MeZO requires a high number of iterations to achieve a good fine-tuning performance and works only in settings where the optimization trajectory is sufficiently well-behaved, i.e. when fine-tuning is coupled with appropriately crafted task prompts. As such, we revisit ZO optimization under the standard (non-prompted) fine-tuning setting. Through empirical studies, we probed further and

identified that the method also contends with i) instability for smaller batch sizes, and ii) a notable convergence gap to first-order (FO) fine-tuning methods in non-prompted settings (see Figures 3.1a, 3.1b, 3.1c).

In this chapter, we demonstrate that variance-reduction enhances the stability and convergence properties of ZO methods in the large-scale LM fine-tuning setting. Based on our observation that ZO methods benefit from improved stability with larger batch sizes, we propose the Memory Efficient Zeroth-Order Stochastic Variance-Reduced Gradient (MeZO-SVRG) method: a ZO algorithm that combines fullbatch and minibatch information to yield asymptotically unbiased, low-variance gradient estimators. Our specific contributions are enumerated below.

1. We perform empirical studies across a range of problem scales to investigate the potential limitations of MeZO. We identified its susceptibility to unstable behavior for smaller batch sizes and convergence issues in spurious optimization landscapes as improvement avenues.

2. We propose MeZO-SVRG: an efficient variant of the ZO-SVRG method that uses in-place operations to achieve a minimal memory footprint and leverages gradient estimators computed with single perturbation vectors to exploit data parallelism for speed.

3. We fine-tune masked and autoregressive LMs (model scales up to 7B) on GLUE [37] and SuperGLUE [38] tasks. MeZO-SVRG achieves consistent performance improvements with up to 20% increase in test accuracies over MeZO across all models and tasks. MeZO-SVRG achieves superior performance to MeZO in both full- and partial-parameter fine-tuning, in both full (FP32) and half (BF16) precision and under standard non-prompt settings.

4. MeZO-SVRG stands out by consistently surpassing MeZO's test accuracy in only half as many GPU-hours.

5. We show that MeZO-SVRG significantly reduces the required memory footprint compared to first-order methods, i.e. by at least $2\times$ for considered autoregressive models. Furthermore, our experiments highlight that MeZO-SVRG's memory savings progressively improve compared to SGD with larger batch sizes.

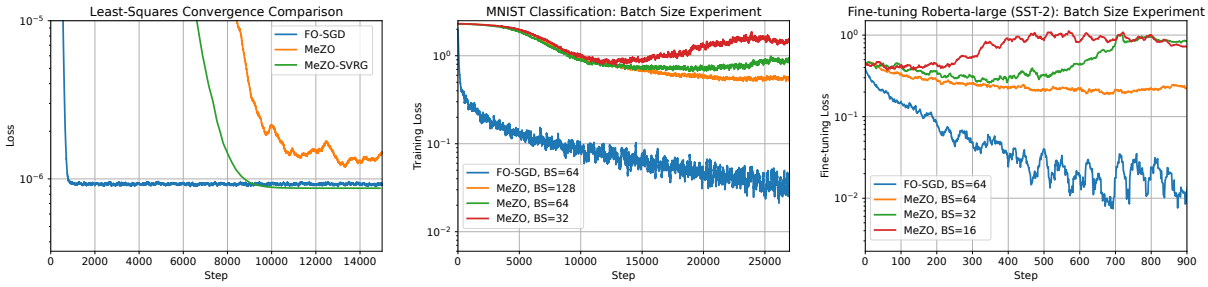
6. We establish convergence guarantees for MeZO-SVRG when equipped with gradient estimators that are computed using single perturbation vectors.

3.2 Background

Zeroth-Order Gradient Estimators

Consider solving the unconstrained optimization

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^d} f(\boldsymbol{\theta}) := \frac{1}{n}\sum_{i=1}^{n} f_i(\boldsymbol{\theta}), \tag{3.1}$$



(a) (b) (c)

Figure 3.1: (a) Shows that MeZO [34] is unable to attain the optimal value when solving least-squares (LS) problems unlike our proposed MeZO-SVRG. In (b) and (c), MeZO is used for MNIST [39] classification and fine-tuning RoBERTa-large on SST-2 [40], respectively, with varying batch sizes. These illustrate MeZO's instability w.r.t. smaller batch sizes.

where $f : \mathbb{R}^d \to \mathbb{R}$ is a non-convex objective. Note that (3.1) is akin to the standard empirical risk minimization framework, where each f_i is the objective evaluated for one of n training samples. For an iterative ZO algorithm, we need to find a means to approximate the gradient. We can define the following stochastic perturbation simultaneous approximation (SPSA) gradient estimator [35]:

$$\hat{\nabla} f_i(\boldsymbol{\theta}) := \frac{f_i(\boldsymbol{\theta} + \mu \mathbf{z}_i) - f_i(\boldsymbol{\theta} - \mu \mathbf{z}_i)}{2\mu} \mathbf{z}_i \text{ for } i \in [n], \tag{3.2}$$

where $\hat{\nabla}$ denotes a gradient estimator, $\mathbf{z}_i \in \mathbb{R}^d$ is a random vector sampled from a standard normal distribution, and $\mu > 0$ is a perturbation scalar. The extension p-SPSA computes the average of p distinct SPSA estimates. Throughout this work, we consider the default setting of $p = 1$ as we didn't observe empirical benefits of setting $p > 1$. The SPSA gradient estimate is an asymptotically unbiased estimator of the true gradient as $\mu \to 0$ when each component in $\mathbf{z}_i$ is mutually independent and zero-mean [35].

Now suppose we have a minibatch $\mathcal{I} \subset [n]$ of size b. This allows us to define the following:

$$\hat{\nabla} f_{\mathcal{I}}(\boldsymbol{\theta}) := \frac{1}{b} \sum_{i \in \mathcal{I}} \hat{\nabla} f_i(\boldsymbol{\theta}), \tag{3.3}$$

and by extension,

$$\hat{\nabla} f(\boldsymbol{\theta}) := \hat{\nabla} f_{[n]}(\boldsymbol{\theta}). \tag{3.4}$$

Observe that the gradient estimator in (3.3) requires $2b$ function queries and sampling b random vectors. In practice, there are two strategies to compute estimators (3.3) and (3.4): accumulate the minibatch estimator in-place by sequentially computing each samplewise estimator, or parallelize the operation by computing the samplewise estimators simultaneously. The trade-off between

the two strategies is that the former has a minimal memory footprint (scales with dimension of problem) but takes longer, while the latter effectively parallelizes the operation but has to store b vectors.

Thus, we define another set of ZO gradient estimators that accommodate data parallelism: we perturb each samplewise SPSA estimator in the same direction $\mathbf{z} \in \mathbb{R}^d$. For minibatch $\mathcal{I} \subset [n]$ of size b we can construct

$$\bar{\nabla} f_{\mathcal{I}}(\boldsymbol{\theta}) := \frac{\frac{1}{b}\sum_{i\in\mathcal{I}}[f_i(\boldsymbol{\theta}+\mu\mathbf{z}) - f_i(\boldsymbol{\theta}-\mu\mathbf{z})]}{2\mu}\mathbf{z}, \tag{3.5}$$

and

$$\bar{\nabla} f(\boldsymbol{\theta}) := \bar{\nabla} f_{[n]}(\boldsymbol{\theta}). \tag{3.6}$$

From an implementation standpoint, estimators (3.5) and (3.6) can exploit data parallelism across the batch $\mathcal{I}$ and benefit from a minimal required memory footprint.

Memory-efficient ZO-SGD (MeZO)

In [34], the authors propose a memory-efficient ZO-SGD optimizer (MeZO) to fine-tune LMs. MeZO is a ZO-SGD algorithm that estimates gradients based on the two-point SPSA estimator introduced in (3.5).

Definition 3. *(ZO-SGD)* Consider solving optimization (3.1). ZO-SGD is an iterative ZO optimizer characterized with update rule

$$\boldsymbol{\theta}^{(t+1)} := \boldsymbol{\theta}^{(t)} - \eta\bar{\nabla} f_{\mathcal{I}}(\boldsymbol{\theta}^{(t)}),$$

for learning rate $\eta > 0$, and SPSA estimator $\bar{\nabla} f_{\mathcal{I}}(\boldsymbol{\theta}^{(t)})$ over minibatch $\mathcal{I} \in [n]$.

Implementing a vanilla ZO-SGD algorithm requires twice the memory footprint of inference due to the need to store the perturbation vector $\mathbf{z} \in \mathbb{R}^d$. In [34], an in-place implementation of the algorithm is proposed, where the requirement of storing a full set of perturbation scalars is mitigated by merely storing a single random seed and regenerating the perturbation vector when required. This brings the memory cost of MeZO down to that of inference (see Section 3.12 for more details on the implementation).

ZO-SVRG

The Zeroth-Order Stochastic Variance Reduced Gradient (ZO-SVRG) [41] method periodically combines a fullbatch gradient estimator with the minibatch estimator to mitigate the stochasticity of the latter. This variance reduction helps achieve a faster convergence rate compared to ZO-SGD [41]. While the full algorithm is presented in Section 3.10, the update rule is:

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \eta[\hat{\nabla} f_{\mathcal{I}_t}(\boldsymbol{\theta}^{(t)}) - \hat{\nabla} f_{\mathcal{I}_t}(\bar{\boldsymbol{\theta}}) + \hat{\nabla} f(\bar{\boldsymbol{\theta}})] \tag{3.7}$$

where $\eta > 0$ is the learning rate, $\mathcal{I}_t$ is a minibatch sampled at iteration t, $\boldsymbol{\theta}^{(t)}$ is the parameter state at iteration t, and $\bar{\boldsymbol{\theta}}$ is the last parameter state at which the fullbatch gradient estimator was computed. Throughout this work, we let $q \in \mathbb{N}$ denote the regularity of fullbatch SPSA computations, i.e. every q steps the fullbatch SPSA estimator is computed.

3.3 Our proposed method: MeZO-SVRG

In this section, we describe the proposed MeZO-SVRG method. We first motivate our method by discussing the observed limitations of MeZO and outline practical implementation concerns when using ZO-SVRG [41] to mitigate these. We then introduce MeZO-SVRG as a variant of ZO-SVRG that minimizes memory usage with in-place operations and accommodates data parallelism in its gradient estimators.

MeZO Limitations

In [34], authors mention that MeZO requires a suitable task prompt to perform well; under this setting the optimization trajectory is more well-behaved. This suggests that the applicability of MeZO is restricted to settings where the optimization landscape is sufficiently well-behaved and cannot be extended to more complex tasks such as pre-training. Moreover, the careful design of prompts for real-world fine-tuning tasks also demands additional effort and may not always be practical. This motivates developing a method that delivers robust performance independently of any reliance on input prompts.

While MeZO has demonstrated promise in fine-tuning settings, our empirical findings suggest that it still faces the following challenges: i) it is susceptible to instability when using smaller batch sizes, and ii) a considerable performance gap with respect to first-order (FO) fine-tuning exists in the non-prompted setting. We illustrate these issues in Figures 3.1a, 3.1b and 3.1c. The details of the experiments are provided in Appendix 3.9. These observations motivate using variance-reduction techniques that leverage larger batch information to improve stability and convergence of ZO methods in the large-scale problem settings.

ZO-SVRG Implementation Concerns

Memory Footprint. Recalling $\boldsymbol{\theta} \in \mathbb{R}^d$, the ZO-SVRG method has a minimum memory requirement of storing d values. A naive implementation of ZO-SVRG presented in Algorithm 4 (see Appendix 3.10) would require an additional $2d$ of memory space for storing the fullbatch gradient estimator and parameter state $\bar{\boldsymbol{\theta}}$ used in (3.7). Moreover, computing and storing $\hat{\nabla} f_{\mathcal{I}_t}(\boldsymbol{\theta}^{(t)})$ and $\hat{\nabla} f_{\mathcal{I}_t}(\bar{\boldsymbol{\theta}})$ also accrues an additional d values of memory each. Thus, a naive implementation of Algorithm 4 would require a minimum memory budget equivalent to $5\times$ the memory budget of inference, which is prohibitive for sufficiently large d.

Iteration Speed Concerns. The original ZO-SVRG method is proposed with the inefficient gradient estimators introduced in (3.3) and (3.4). In both, SPSA estimators are computed for

individual samples and averaged over the batch. Consider computing (3.3) with batch size b. If we want to fully parallelize operations, we require computing and storing b many $\hat{\nabla} f_i(u)$ estimators. However, this increases the memory footprint. To save on memory usage, in-place operations can be used. However, this has the effect of drastically reducing the computation speed as we need to sequentially compute each of the b estimators in (3.3).

MeZO-SVRG

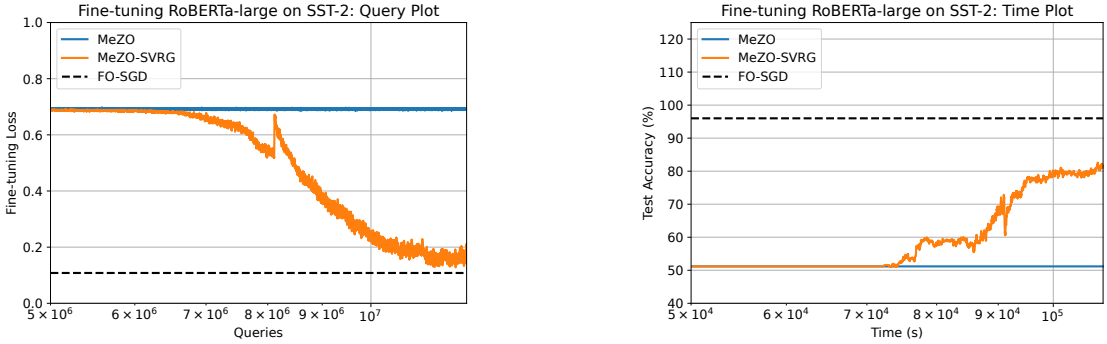
We propose MeZO-SVRG: a variant of ZO-SVRG that improves iteration speed by using estimators (3.5), (3.6) and reduces the memory footprint with in-place operations. The method is summarized in Algorithm 3.

Efficient Gradient Estimation. We utilize the efficient gradient estimators introduced in (3.5) and (3.6) that perturb the entire batch in a single direction. These estimators accommodate data parallelism offered by modern ML frameworks. Furthermore, we can utilize the "resampling trick" introduced in [34] to reduce the memory footprint when computing each of (3.5) and (3.6); each estimator requires a memory footprint equivalent to the problem dimension d (see Appendix 3.12 for the memory-efficient SPSA computation procedure). Thus, using estimators (3.5) and (3.6), eliminates the memory/speed trade-off plaguing the ZO-SVRG implementation and get the best of both worlds.

Algorithm 3 Memory-Efficient ZO-SVRG (MeZO-SVRG)

Input: Total iterations T, learning rates $\eta_1, \eta_2 > 0$, minibatch size b, parameters $\boldsymbol{\theta}_0$, iterations between full-batch gradient $q \in \mathbb{N}$
begin method
for $t = 0, \ldots, T$ **do**
  if $t \mod q = 0$ **then**
    1. $\mathbf{g} \leftarrow \bar{\nabla} f(\boldsymbol{\theta}^{(t)})$
    2. $\bar{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}^{(t)}$
    3. update: $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \eta_1 \mathbf{g}$ `#in-place`
  else
    4. Choose mini-batch $\mathcal{I}_t$ of size b
    5. $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t)} - \eta_2 \bar{\nabla} f_{\mathcal{I}_t}(\boldsymbol{\theta}^{(t)})$ `#in-place`
    6. $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t)} + \eta_2 \bar{\nabla} f_{\mathcal{I}_t}(\bar{\boldsymbol{\theta}})$ `#in-place`
    7. update: $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \eta_2 \mathbf{g}$ `#in-place`
  end if
end for
end

In-place Operations for Memory Efficiency. MeZO-SVRG leverages in-place operations to minimize memory allocation for new variable definitions. Memory space is required for the current state of the d parameters, a copy of the parameter state after each fullbatch SPSA computation as well as the fullbatch SPSA estimator itself. This requires a minimum memory requirement of storing $3d$ values. The minibatch updates can then be computed in-place in Lines 5, 6, and 7; thus, MeZO-SVRG achieves a reduced minimum memory footprint to $3\times$ that of inference.



(a) (b)

Figure 3.2: Performance of MeZO-SVRG, MeZO and FO-SGD when fine-tuning RoBERTa-large on the SST-2 [40] dataset. The dashed line serves as a reference to the training loss/test accuracy achieved by FO-SGD. (a) MeZO-SVRG is able to significantly reduce the convergence gap to FO-SGD compared to MeZO. (b) MeZO-SVRG attains a considerably better test accuracy than MeZO.

Remark 7. As MeZO-SVRG queries the loss function with an inference pass through a network, it minimizes the storage of activations and intermediate variables. The memory footprint of MeZO-SVRG thus mainly stems from retaining copies of the fullbatch gradient estimator and parameters. Therefore, this method scales well with increasing batch sizes. Table 3.3 shows that for increasing batch sizes of up to 64, MeZO-SVRG yields more than 70% memory savings compared to first-order SGD (FO-SGD) on the RoBERTa-large [42] model. Similarly, MeZO-SVRG improves significantly on memory usage compared to FO-SGD for large context lengths and a fixed batch size (consistently $2\times$ smaller footprint, see Figure 3.3).

Remark 8. By storing $\bar{\boldsymbol{\theta}}$ in Line 2 (Algorithm 3), we can keep recomputing the fullbatch estimator on demand without storing $\mathbf{g}$. This would lower the memory footprint of MeZO-SVRG to $2\times$ that of inference. However, as computing the fullbatch estimator can slow down the iteration speed, throughout our experiments we store it.

Remark 9. The memory analysis above does not account for any constant implementation overhead or intermediate activation storage during a forward pass of a network. Thus, in practice the memory usage ratio between MeZO and MeZO-SVRG is smaller (see Table 3.3).

Remark 10. In practice, fine-tuning datasets can be large enough that computing fullbatch SPSA estimators is infeasible (e.g. more than 10^5 training examples). MeZO-SVRG can be adapted so that the fullbatch estimator is approximated with a large batch estimator (e.g. with 512 or 1024 samples). In this case, the updates blend minibatch and large batch (as opposed to fullbatch) information.

Additional Learning Rate. In Algorithm 3, we also include two independent learning rates η_1 and η_2 for the fullbatch and minibatch updates as shown in Lines 3 and Lines 5-7, respectively, of Algorithm 3. This design choice is based on our empirical observation that fullbatch updates are more accommodating of larger learning rates than minibatch steps. In our experiments we find that setting $\eta_1 > \eta_2$ improves convergence speed (see Appendices 3.13, 3.14, 3.15).

	DistilBert Full-Precision (FP32)				**RoBERTa-large** Full-Precision (FP32)			
Method	MNLI	QNLI	SST-2	CoLA	MNLI	QNLI	SST-2	CoLA
MeZO (Full FT)	36 (1.09)	50 (0.69)	52 (0.68)	63 (0.64)	43 (0.94)	59 (0.58)	56 (0.69)	68 (0.51)
MeZO-SVRG (Full FT)	**46 (0.08)**	**68 (0.23)**	**72 (0.02)**	**68 (0.28)**	**49 (0.81)**	**80 (0.28)**	**84 (0.13)**	**79 (0.06)**
FO-SGD (Full FT)	59 (0.01)	78 (0.04)	88 (0.01)	70 (0.02)	85 (0.03)	89 (0.01)	96 (0.11)	85 (0.01)
MeZO (Partial FT)	35 (1.09)	52 (0.69)	51 (0.70)	60 (0.64)	42 (1.07)	50 (0.69)	54 (0.68)	65 (0.59)
MeZO-SVRG (Partial FT)	**47 (0.28)**	**65 (0.29)**	**74 (0.10)**	**67 (0.36)**	**43 (0.82)**	**67 (0.46)**	**72 (0.59)**	**79 (0.35)**
FO-SGD (Partial FT)	48 (0.26)	59 (0.42)	85 (0.05)	66 (0.45)	52 (0.99)	72 (0.60)	89 (0.58)	84 (0.41)

Table 3.1: Experiments on DistilBert and RoBERTa-large. We show the test accuracies and fine-tuning losses (in parentheses) of MeZO-SVRG and MeZO for both full/partial-parameter FT. We also provide results for FO-SGD as an upper-bound benchmark on performance. MeZO-SVRG consistently outperforms MeZO and significantly closes the gap to FO-SGD. Detailed results are given in Tables 3.12 and 3.14.

	GPT2 Full-Precision (FP32)			**OPT-2.7B** Full-Precision (FP32)			**OPT-6.7B** Half-Precision (BF16)	
Method	MNLI	SST-2	CoLA	MNLI	SST-2	CoLA	SST-2	BoolQ
MeZO	41 (0.65)	59 (0.32)	61 (0.35)	42 (1.09)	61 (0.65)	62 (0.58)	74 (0.53)	65 (0.63)
MeZO-SVRG	**53 (0.41)**	**65 (0.20)**	**69 (0.25)**	**52 (0.81)**	**65 (0.55)**	**67 (0.53)**	**77 (0.52)**	**69 (0.57)**
FO-SGD	69 (0.59)	72 (0.23)	78 (0.38)	78 (0.33)	98 (0.02)	94 (0.17)	91 (0.10)	84 (0.29)

Table 3.2: Experiments on AR models. We show the test accuracies and fine-tuning losses (in parentheses) of MeZO-SVRG and MeZO for full-parameter FT. For reference we also provide results for FO-SGD as an upper-bound benchmark on performance. MeZO-SVRG consistently outperforms MeZO and approaches FO-SGD performance. Detailed results are found in Tables 3.17, 3.18 and 3.24.

Storage Efficiency of MeZO-SVRG. Parameter-efficient fine-tuning (PEFT) reduces the size of fine-tuned model checkpoints by optimizing only a small subset of parameters, e.g. LoRA [43] and prefix-tuning [44]. Both MeZO and MeZO-SVRG have the benefit of being able to recover an entire fine-tuning trajectory by storing a single seed and the difference of loss scalars in (3.5) at each step. The stored seed can regenerate step-wise seeds to recover the perturbation vectors $\mathbf{z}$ used

	Largest OPT/GPT that can fit		Memory Usage in GB for RoBERTa-large				
			Fixed context length (cl=128)			Fixed batch size (bs=64)	
Method	A100 (40GB)	H100 (80GB)	bs $= 16$	bs $= 32$	bs $= 64$	cl $= 256$	cl $= 512$
MeZO	6.7B	13B	2.07 (69%)	2.21 (79%)	2.51 (88%)	3.35	5.97
MeZO-SVRG	**2.7B**	**6.7B**	**4.36 (35%)**	**4.51 (58%)**	**4.72 (76%)**	5.13	8.02
FO-SGD	1.6B	2.7B	6.74	10.67	18.55	OOM	OOM
FO-Adam	350M	1.3B	10.44	14.33	22.41	OOM	OOM

Table 3.3: Shows the largest AR models that can fit on single 40, 80GB GPUs. We also measure the memory usage under different batch sizes (bs) and context lengths (cl) when fine-tuning RoBERTa-large. Percentages indicate the memory savings with respect to FO-SGD.

for each SPSA computation. Together with the stored difference in loss values, we can recover the exact gradient estimators used in the fine-tuning process without needing to perform any forward passes. This allows recovering any model checkpoint along the fine-tuning trajectory. As we store only the initial random seed and a sequence of difference of loss scalars, we can achieve significant storage efficiency.

Compatibility with Non-differentiable Objectives and PEFT. As MeZO-SVRG uses only forward passes and a difference of loss values to estimate the gradient, it is applicable to settings where gradients are inaccessible or infeasible to compute, e.g. when considering non-differentiable objectives such as ranking in RLHF [45] or access to model gradients is restricted. Similar to MeZO, MeZO-SVRG also remains compatible with PEFT (e.g. LoRA [43], prefix-tuning [44]).

3.4 Experiments

In this section, we evaluate MeZO-SVRG on a variety of fine-tuning tasks by comparing the performance against MeZO [34] and memory usage against first-order stochastic gradient descent (FO-SGD) [31] and first-order Adam (FO-Adam) [32]. We demonstrate empirically that MeZO-SVRG performs well in the absence of input prompts: it is able to significantly reduce the performance gap to FO methods and consistently surpasses MeZO's performance on a variety of fine-tuning tasks with significantly lower computation time. Furthermore, MeZO-SVRG necessitates a considerably smaller memory footprint compared to FO-SGD and FO-Adam.

Setup. We evaluate on both full (FP32) and half (BF16) precision. We detail the experiment results for the BF16 setting in Section 3.17.We mainly consider a prompt-free fine-tuning setting (more challenging loss landscape) but include prompted results for RoBERTa-large [42] in Section 3.14. All experiments are run on a single GPU; specifically, we consider Nvidia A100 40GB or H100 80GB GPUs. We evaluate the algorithms under two fine-tuning strategies: full- and partial-parameter fine-tuning. In the latter we fine-tune the last layers of the chosen models. We define a query as one forward pass for a single sample. For a fair comparison between MeZO and

MeZO-SVRG, we ensured that the total number of queries used by both remains the same; thus, as MeZO-SVRG accrues more queries per step due to the fullbatch gradient estimates, MeZO was run for more steps. Further details of the experiment setup and implementation are provided in Sections 3.11 and 3.12.

Dataset. We fine-tune on tasks from the NLP GLUE and SuperGLUE benchmarks: Multi-Genre Natural Language Inference Corpus (MNLI), Stanford Question Answering Dataset (QNLI), Stanford Sentiment Treebank (SST-2), Corpus of Linguistic Acceptability (CoLA), and BoolQ [37], [38], [40], [46], [47]. Similar to [34], for each task, our experiments are conducted in a many-shot fine-tuning setting: 512 training examples, 256 validation examples and 256 test samples are randomly sampled from the dataset.

Language Models. We considered Distilbert [48] and RoBERTa-large as our masked LMs. Details on the hyperparameter configuration used for these experiments are provided in Section 3.13, 3.14 and 3.17. We extend our evaluation to fine-tuning larger autoregressive (AR) models. We consider the GPT2 [49], OPT-2.7B, and OPT-6.7B [50] models. The hyperparameter configurations used for these experiments are detailed in Section 3.15 and 3.17.

LM Fine-tuning Performance

	GPT2				OPT-2.7B			
Method	MNLI	QNLI	SST-2	CoLA	MNLI	QNLI	SST-2	CoLA
MeZO	0.4	5.5	19.4	2.8	2.6	5.3	48	55
MeZO-SVRG	**0.3**	**1.9**	**5.6**	**2.2**	**1.1**	**2.7**	**25**	**1.4**

Table 3.4: Required GPU-hrs to achieve equivalent performance levels for MeZO-SVRG and MeZO.

MeZO-SVRG significantly outperforms MeZO in both the fine-tuning loss convergence and test accuracy. On all models and tasks, MeZO-SVRG improves on the test accuracy over MeZO: we see an improvement of up to 20% in Tables 3.1, 3.2 and Figure 3.2b. MeZO-SVRG also consistently achieves an improved fine-tuning loss compared to MeZO. This is particularly evident in Figure 3.2a. Additional results are presented in Sections 3.13, 3.14 and 3.15.

MeZO-SVRG works well on both full and partial fine-tuning. The improvement over MeZO is consistent across both fine-tuning modes. In partial fine-tuning, MeZO-SVRG often achieves comparable performance to FO-SGD (within 5%) on several tasks (see Table 3.1).

MeZO-SVRG closes the gap to FO-SGD in training convergence and matches the test accuracy. Tables 3.1 and 3.2 demonstrate how MeZO-SVRG closes the performance gap with FO-SGD compared to MeZO.

MeZO-SVRG's superior performance to MeZO extends to the low (half) precision (BF16) setting. We summarize the half-precision results in Section 3.17.

Memory Usage Profiling

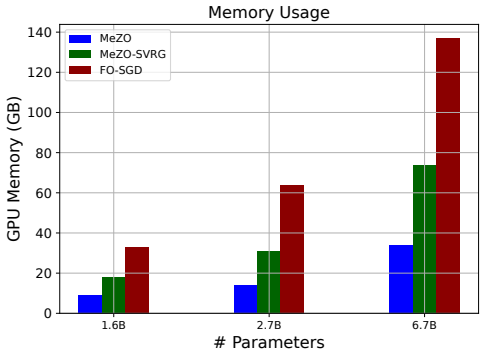


Figure 3.3: Shows the minimum memory usage on autoregressive models (batch size $= 1$, use max context length of model). MeZO-SVRG yields a $2\times$ smaller memory footprint over FO-SGD.

MeZO-SVRG can fit larger models on the same hardware than FO-SGD. We measure the minimum memory requirement to fine-tune (full-parameter) the considered autoregressive models using the different methods. We fine-tune GPT2, OPT-2.7B and OPT-6.7B on MNLI by setting the input sequence length to the maximum context length of the LM and report the peak GPU memory consumption for batch size $= 1$. Table 3.3 shows that *MeZO-SVRG consistently yields a significantly improved memory footprint compared to FO-SGD (approximately* $2\times$ *across considered autoregressive models).* More details on how memory profiling was done is summarized in Section 3.16.

MeZO-SVRG's memory savings progressively improve over FO-SGD and FO-Adam with increasing batch size and context lengths. For this experiment, we consider the masked model RoBERTa-large. Again we fine-tune on the MNLI dataset using a single Nvidia A100 40GB GPU and set the input sequence length to a constant size of 128. We measure the peak GPU memory consumption for the different methods for varying batch sizes $\{16, 32, 64\}$. Figure 3.3 shows that for a fixed model (RoBERTa-large) and context length (128), MeZO-SVRG exhibits memory savings of up to 76% w.r.t FO-SGD. We also vary the context lengths $\{256, 512\}$ of the input for a fixed batch size (64). Again we observe significant benefits for MeZO-SVRG over FO-SGD: the latter is subject to out-of-memory errors when running this setting with 40GB GPUs.

MeZO-SVRG consumes more memory than MeZO due to its need to store copies of the parameters and fullbatch SPSA estimators (see Algorithm 9), but compensates by delivering notable gains in test performance and computation time.

Computation Time

We compare the speed of MeZO-SVRG and MeZO by measuring the total GPU-hours required to achieve a certain performance threshold. For a fair comparison, we set the threshold to a level attained by both methods, namely, MeZO's peak test accuracy. Table 3.4 shows that for GPT2

and OPT-2.7B, MeZO-SVRG consistently achieves superior test accuracy with less than half the GPU-hours.

Understanding MeZO-SVRG

To better understand how the perturbation scale μ and regularity of full-batch update steps determined by q impact the MeZO-SVRG performance, we perform ablation studies in Section 3.12 with DistilBert on the MNLI dataset. For large fine-tuning datasets, estimating the full-batch gradient can be impractical. Therefore, we included an ablation study to examine the impact on MeZO-SVRG performance when substituting the full-batch gradient estimator with a large-batch estimator. Results in Table 3.10 suggest that large-batch estimators pose an effective alternative to full-batch estimators.

3.5 Convergence Theory

In this section, we provide a convergence analysis of MeZO-SVRG. We start by showing that our estimator is unbiased w.r.t. a minibatch set $\mathcal{I}$. We assume $\mathcal{I}$ is drawn either uniformly random with or without replacement.

Lemma 1.

$$\mathbb{E}_{\mathcal{I}}\bar{\nabla} f_{\mathcal{I}}(\boldsymbol{\theta}) = \bar{\nabla} f(\boldsymbol{\theta}) \tag{3.8}$$

We denote $\mathbf{u}_{\mathcal{I}} = \bar{\nabla} f_{\mathcal{I}}(\boldsymbol{\theta}) - \bar{\nabla} f_{\mathcal{I}}(\boldsymbol{\theta}') - \mathbb{E}_{\mathcal{I}}[\bar{\nabla} f_{\mathcal{I}}(\boldsymbol{\theta}) - \bar{\nabla} f_{\mathcal{I}}(\boldsymbol{\theta}')]$ and $\mathbf{u}_{\mathcal{I}} = \mathbf{u}_i$ for $\mathcal{I} = \{i\}$. This $\mathbf{u}_i$ is a key component from the idea of control covariates [51] in reducing variance.

Lemma 2. $\sum_{i=1}^{n} \mathbf{u}_i = 0$ *and* $\mathbb{E}_{\mathcal{I}}[\mathbf{u}_i \mathbf{u}_j] = 0$ *where* $i, j \in \mathcal{I}$ *and* $i \neq j$.

Assumptions. A1: Functions $\{f_i\}$ are L-smooth, i.e., $\|\nabla f_i(\boldsymbol{\theta}) - \nabla f_i(\boldsymbol{\theta}')\| \leq L\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|_2^2$. A2: The variance of stochastic gradients is bounded as $\frac{1}{n}\sum_{i=1}^{n} \|f_i(\boldsymbol{\theta}) - f(\boldsymbol{\theta}')\|_2^2 \leq \sigma^2$.

With the two Lemmas and assumptions, the following holds.

Jibberish 4. Assume A1 and A2 holds. Let learning rates $\eta = \eta_1 = \eta_2$. Then, MeZO-SVRG satisfies

$$\mathbb{E}[\|\nabla f(\boldsymbol{\theta}^{(T)})\|_2^2] \leq \frac{f(\boldsymbol{\theta}^{(0)}) - f^*}{T\bar{\gamma}} + \frac{L\mu^2}{T\bar{\gamma}} + \frac{c}{q\bar{\gamma}} \tag{3.9}$$

where $\bar{\gamma}$ and c are functions of learning rate η, dimension d, minibatch size b and L, σ. Moreover, by setting

$$\mu = \frac{1}{\sqrt{dT}}, \quad \eta = \frac{\rho}{L}, \quad q = \lceil \frac{d}{31\rho} \rceil,$$

where ρ is a universal constant, MeZO-SVRG satisfies

$$\mathbb{E}[\|\nabla f(\boldsymbol{\theta}^{(T)})\|_2^2] = O\left(\frac{d}{T} + \frac{\mathbf{1}(b < n)}{b}\right). \tag{3.10}$$

Theorem 4 demonstrates a linear convergence, inverse proportional to iteration T and q. The second term in Eq. (3.9) expresses the effect of μ, the magnitude of perturbation, which is small in practice. In Eq. (3.10), q is proportional to the problem dimension d, which can balance overall computational cost. It also reveals the effect of batch size b, indicating larger batch sizes are preferred in terms of iteration counts, which coincide with our empirical observation.

Remark 11. The derivation of Theorem 4 heavily relies on mathematical machinery and flows of original SVRG [52] and ZO-SVRG [41]. However, note the gradient estimators in MeZO-SVRG and ZO-SVRG are different, e.g. different scaling, a single perturbation vector $\mathbf{z}$ vs multiple perturbation vectors $\{\mathbf{z}_i\}$ against `RandGradEst` [41]. This requires careful examination often with different derivation like Lemma 1, 2, while making sure random vector $\mathbf{z}$ is conditioned consistently over sequence of derivations in [41]. A proof sketch clarifying main distinct steps is provided in Appendix 3.8.

3.6 Related work

Zeroth-Order Optimization. Zeroth-order (ZO) methods solve optimization problems without using gradient information. This class of methods typically estimates the gradient from function queries. Convergence theory has been developed for ZO stochastic gradient descent (ZO-SGD) in both convex [53]–[55] and non-convex settings [41], [56]. However, these bounds generally depend on the number of parameters d. In [34], authors demonstrate via fine-tuning experiments that after pre-training and the inclusion of task prompts, the loss landscape is well-behaved enough and can be traversed by ZO-SGD. [57] benchmarks the performance of ZO methods in the context of LM fine-tuning. However, to the best of our knowledge, we are the first to explore the direction of variance-reduced ZO optimization for fine-tuning LMs.

Memory-efficient Backpropagation Strategies. LLMs are typically fine-tuned by using FO methods such as SGD [31] and Adam [32]. Several methods have been proposed to handle the memory overheads of backpropagation, for e.g. sparsifying gradients [58], [59] and quantizing gradients to lower bit precisions [60], [61]. Other techniques to save activation memory during forward and backward pass include Gradient checkpointing [62] and Flash Attention [63].

Gradient-free Adaptation of LLMs. The pre-training stage gives LLMs the ability to generalize to tasks for which it has not been explicitly trained. This form of adaptation requires instruction prompts and is referred to as *in-context learning* (ICL). While ICL enables quick adaptation of the model to specific tasks, drawbacks of this approach include that current models are constrained to limited context window and are sensitive to both the choice of input prompts and demonstrations [34]. Moreover, it has been empirically demonstrated that ICL on large models performs worse than full fine-tuning on medium-scale models [64]. Another paradigm of adapting LLMs without using gradients is by using evolutionary algorithms [65], [66], however the effectiveness of these methods has not been verified beyond smaller LMs.

3.7 Conclusion

This chapter introduces MeZO-SVRG: a variance-reduced ZO method that addresses the challenge of fine-tuning LLMs under memory constraints. MeZO-SVRG is a variant of ZO-SVRG that exploits in-place operations for memory-frugality and gradient estimators that accommodate data parallelism for iteration speed. The method combines fullbatch and minibatch information to yield low variance gradient estimators. We demonstrate empirically that MeZO-SVRG outperforms MeZO consistently on a variety of LM fine-tuning tasks, even in a challenging non-prompted setting, and requires significantly less GPU-hours to achieve this performance. Furthermore, we show that across model types and fine-tuning tasks, MeZO-SVRG is able to considerably close the performance gap to SGD while benefiting from a $2\times$ reduction in memory utilization.

We are excited to further explore the potential of MeZO-SVRG. In particular, we aim to examine MeZO-SVRG's performance when coupled with PEFT (LoRA, prefix-tuning) and settings where gradient-information is unavailable, e.g. prompt-tuning black-box models that are accessible only through an API. Finally, our work paves the way for exploring a broader spectrum of variance reduction techniques for ZO methods in the context of LM fine-tuning.

3.8 Additional Results: Proof of Theorem

Throughout the proof, we drop bold notations $\boldsymbol{\theta}, \mathbf{z}, \mathbf{u} \to \theta, z, u$ for notational simplicity.

Lemma 3.

$$\mathbb{E}_{\mathcal{I}} \bar{\nabla} f_{\mathcal{I}}(\theta) = \bar{\nabla} f(\theta) \tag{3.11}$$

Proof.

$$\begin{aligned}
\mathbb{E}_{\mathcal{I}} \bar{\nabla} f_{\mathcal{I}}(\theta) &= \frac{1}{b} \mathbb{E}_{\mathcal{I}} \sum_{i \in \mathcal{I}} \frac{f_i(\theta + \mu z) - f_i(\theta - \mu z)}{2\mu} z \\
&= \frac{1}{b} \frac{b}{n} \sum_{i=1}^{n} \frac{f_i(\theta + \mu z) - f_i(\theta - \mu z)}{2\mu} z \\
&= \bar{\nabla} f(\theta)
\end{aligned}$$

The first and third equality comes from the definition of $\bar{\nabla} f_{\mathcal{I}}, \bar{\nabla} f$ and the second equality holds due to re-ordering under the assumption a minibatch set is sampled uniformly random or random with permutation. □

We denote $u_{\mathcal{I}} = \bar{\nabla} f_{\mathcal{I}}(\theta) - \bar{\nabla} f_{\mathcal{I}}(\theta') - \mathbb{E}_{\mathcal{I}}[\bar{\nabla} f_{\mathcal{I}}(\theta) - \bar{\nabla} f_{\mathcal{I}}(\theta')]$ and $u_{\mathcal{I}} = u_i$ for $\mathcal{I} = \{i\}$.

Lemma 4. $\sum_{i=1}^{n} u_i = 0$ *and* $\mathbb{E}_{\mathcal{I}}[u_i u_j] = 0$ *where* $i, j \in \mathcal{I}$ *and* $i \neq j$.

Proof. By definition, $\sum_{i=1}^{n} \bar{\nabla} f_i(\theta) = n \bar{\nabla} f(\theta)$. It is immediate to see $\sum_{i=1}^{n} \mathbb{E}_{\mathcal{I}}[\bar{\nabla} f_{\mathcal{I}}(\theta)] = n \bar{\nabla} f(\theta)$, similar to Lemma 3. Therefore $\sum_{i=1}^{n} u_i = 0$ holds. Conditioned on other randomness, e.g. perturbation z, $\mathbb{E}_{\mathcal{I}}[u_i u_j] = 0$ as i, j are independent. □

Assumptions. A1: Functions $\{f_i\}$ are L-smooth, i.e. $\|\nabla f_i(\theta) - \nabla f_i(\theta')\| \leq L \|\theta - \theta'\|_2^2$. A2: The variance of stochastic gradients is bounded as $\frac{1}{n} \sum_{i=1}^{n} \|f_i(\theta) - f(\theta')\|_2^2 \leq \sigma^2$.

Equipped with two Lemmas, the following holds

Jibberish 5. Assume A1 and A2 holds. Let learning rate $\eta = \eta_1 = \eta_2$. Then, MeZO-SVRG satisfies

$$\mathbb{E}[\|\nabla f(\theta^{(T)})\|_2^2] \leq \frac{f(\theta^{(0)}) - f^*}{T\bar{\gamma}} + \frac{L\mu^2}{T\bar{\gamma}} + \frac{c}{q\bar{\gamma}} \tag{3.12}$$

where $\bar{\gamma}$ and c are the functions of stepsize η, dimension d, mini-batch size b and L, σ. Moreover, by setting

$$\mu = \frac{1}{\sqrt{dT}}, \quad \eta = \frac{\rho}{L}, \quad q = \lceil \frac{d}{31\rho} \rceil$$

where ρ is a universal constant, MeZO-SVRG satisfies

$$\mathbb{E}[\|\nabla f(\theta^{(T)})\|_2^2] = O\left(\frac{d}{T} + \frac{\mathbf{1}(b < n)}{b} \right). \tag{3.13}$$

Proof. We rely on the proof provided by [41]. Note that we need to make sure that certain important steps and Lemmas still hold under MeZO-SVRG's gradient estimators. We start by using $d\bar{\nabla} f$ as our gradient estimate, through which Lemma 1 and 2 (in in [41]) hold by matching the scale of gradient to `RandGradEst` in [41]. Lemma 3 is used for Eq. (36) (Proposition 1 of [41]). Lemma 4 is used for Lemma 4, 5 in [41]. Eq. (40) (Proposition 1 of [41]) holds because of a different conditional expectation, i.e., $E = E_z E_{\mathcal{I}|z} = E_z E_{\mathcal{I}}$, rather than $E = E_{\{z_i\}} E_{\mathcal{I}|\{z_i\}} = E_{\{z_i\}} E_{\mathcal{I}}$ where z and $\{z_i\}$ are random perturbations. The rest of proof follows through algebraic inequalities based on Lemmas 1,2, 4,5, and function assumptions, to derive convergence analysis. Finally we scale down learning rate η by d to adopt the gradient estimate of our definition. □

3.9 Additional Results: Exploring the Limits of MeZO Empirically

MNIST classification and RoBERTa-large fine-tuning

We ran experiments to better understand shortcomings in MeZO [34]. Two settings were considered: performing MNIST [39] classification with a two-layer MLP (25K parameters) and fine-tuning RoBERTa-large (350M parameters) on the SST-2 [40] dataset. In the former, we used a two-layer feedforward network with 32 and 16 hidden units respectively. In the latter, we performed full-parameter fine-tuning. In [34], authors also remark that a simple instruction prompt is needed for the algorithm to succeed in fine-tuning tasks, i.e. it requires a sufficiently well-behaved optimization trajectory. While this, in itself, can be noted as a drawback, we adopted their proposed prompts in the experiment [34]. The training and fine-tuning runs are illustrated in Figures 3.1b and 3.1c. The hyperparameters selected for the runs are summarized in Tables 3.5 and 3.6. We paid particular attention to the effect of varying batch size on the algorithm performance. We also varied the perturbation scale μ used in the SPSA estimates (3.5). No improvement was found in reducing μ from the default setting used in MeZO ($\mu = 1e-3$) and thus we present results only for that configuration [34]. The largest learning rate values used in the grid search were selected for the MeZO runs. As an upper bound reference on performance, we also include the training curves for the FO-SGD algorithm. From both Figures 3.1b and 3.1c, it is clear the MeZO has to contend with instability incurred at smaller batch sizes.

Solving Least Squares

To make the aforementioned observations even more apparent, we examined the performance of MeZO on a simple linear least-squares (LS) problem. Specifically we solve

$$\min_{w \in \mathbb{R}^d} \|Xw - y\|_2^2, \tag{3.14}$$

where $X \in \mathbb{R}^{n \times d}$ is a randomly generated matrix, $w \in \mathbb{R}^d$ is fixed a priori, and $y \in \mathbb{R}^n = Xw +$ noise is the target labels. In our experiment, we focus on the 100-dimensional problem, i.e.

Table 3.5: The hyperparameter grid optimized over in the initial the small-scale MNIST [39] classification experiments.

Algorithm	**Hyperparameters**	**Values**
MeZO	Batch size	$\{32, 64, 128\}\times$
	Learning rate	$\{1e-3, 1e-4\}\times$
	μ	$\{1e-3, 1e-4, 1e-5\}$
FO-SGD	Batch size	$\{64\}\times$
	Learning rate	$\{1e-3\}$

Table 3.6: The hyperparameter grid optimized over in the initial RoBERTa-large [42] fine-tuning experiments.

Algorithm	**Hyperparameters**	**Values**
MeZO	Batch size	$\{16, 32, 64\}\times$
	Learning rate	$\{1e-5, 1e-6\}\times$
	μ	$\{1e-3, 1e-4, 1e-5\}$
FO-SGD	Batch size	$\{64\}\times$
	Learning rate	$\{1e-5\}$

with $d = 100$ and $n = 1000$. For comparison, we also report the performances of our proposed MeZO-SVRG and FO-SGD. The hyperparameter configurations used are presented in Table 3.7. Figure 3.1a makes it clear that MeZO is unable to attain the optimal value and yields a performance gap w.r.t. MeZO-SVRG and FO-SGD.

Table 3.7: The hyperparameters used for the Least Squares (LS) convergence experiment.

Algorithm	**Hyperparameters**	**Values**
MeZO	Batch size	$\{32\}\times$
	Learning rate	$\{1e-3\}\times$
	μ	$\{1e-3\}$
MeZO-SVRG	Batch size	$\{32\}\times$
	Learning rate (η_1)	$\{1e-3\}\times$
	Learning rate (η_2)	$\{1e-4\}\times$
	μ	$\{1e-3\}\times$
	q	$\{2\}$
FO-SGD	Batch size	$\{32\}\times$
	Learning rate	$\{1e-3\}$

3.10 Additional Results: Zeroth-Order Stochastic Variance-Reduced Gradient

For the sake of completeness, we present the ZO-SVRG algorithm proposed in [41]. This algorithm was proposed without a focus on memory efficiency, in contrast to our MeZO-SVRG, which offers significant memory-saving advantages, particularly in the context of fine-tuning large-scale LMs.

Algorithm 4 ZO-SVRG [41]

Input: Total iterations T, learning rate $\eta > 0$, minibatch size b, parameters $\boldsymbol{\theta}_0$, iterations between fullbatch estimators $q \in \mathbb{N}$
begin method
for $t = 0, \ldots, T$ **do**
  if $t \mod q = 0$ **then**
    1. $\mathbf{g} \leftarrow \hat{\nabla} f(\boldsymbol{\theta}^{(t)})$
    2. $\bar{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}^{(t)}$
  end if
  3. Choose mini-batch $\mathcal{I}_t$ of size b
  4. $\hat{\mathbf{g}} \leftarrow \hat{\nabla} f_{\mathcal{I}_t}(\boldsymbol{\theta}^{(t)})$
  5. $\bar{\mathbf{g}} \leftarrow \hat{\nabla} f_{\mathcal{I}_t}(\bar{\boldsymbol{\theta}})$
  6. Compute gradient blending: $\mathbf{v}_t \leftarrow \hat{\mathbf{g}} - \bar{\mathbf{g}} + \mathbf{g}$
  7. update: $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \eta \mathbf{v}^{(t)}$
end for
end

3.11 Additional Results: Experiment Setup

Datasets

For experiments on LMs, we considered fine-tuning on classification datasets. Specifically, we focused on the following datasets from the General Language Understanding Evaluation (GLUE) [37] benchmark: Multi-Genre Natural Language Inference (MNLI) [46], Question Natural Language Inference (QNLI) [37] for sentence pair classification, Stanford Sentiment Treebank (SST-2) [40] for sentiment analysis, and Corpus of Linguistic Acceptability (CoLA) [47]. To incorporate a more challenging task, we also evaluated on the BoolQ dataset from the SuperGLUE [38] benchmark.

The datasets are imported from the Huggingface `datasets` library. We randomly sampled 512 examples for training, 256 for validation and 256 for testing.

Model

In our implementation, we used models from the Huggingface `transformers` package. As we considered classification datasets, we instantiated models from the `AutoModelsForSequenceClassification` and `OPTModelsForSequenceClassification` classes. These libraries add a classification head on top of the considered pre-trained model. For the prompted experiment setting, we instantiate from the `RobertaModelForPromptFinetuning` custom class implemented in the MeZO repository [34].

Tables 3.8 and 3.9 summarize the models that where considered in our experiments. For the masked models both full- and partial parameter fine-tuning was performed.

Model	Total Trainable Parameters ($\times 10^6$)	Partial Fine-tuning Layers	Partial Fine-tuning Nr. of Parameters ($\times 10^6$)
DistilBert	66	`transformer.layer.5` `classifier`	8

RoBERTa-large	355	`roberta.encoder.layer.20`
		`roberta.encoder.layer.21`
		`roberta.encoder.layer.22`
		`roberta.encoder.layer.23`
		`classifier`
			38

Table 3.8: An overview of the masked LMs used in the experiments. Both full- and partial-parameter fine-tuning was considered for these LLMs.

Model	Total Trainable Parameters ($\times 10^6$)
GPT2 (`gpt2-xl`)	1557
OPT-2.7B (`facebook/opt-2.7B`)	2651
OPT-6.7B (`facebook/opt-6.7B`)	6658

Table 3.9: An overview of the autoregressive LMs used in the experiments.

3.12 Additional Results: MeZO-SVRG Implementation and Ablations

Memory-efficient SPSA

In our implementation we adopt the memory-efficient strategy of computing the SPSA estimator as proposed in [34]. Rather than sampling and storing the entire perturbation vector $\mathbf{z} \in \mathbb{R}^d$, we sample a random seed and use it to regenerate the random vector when required. This allows in-place perturbations of the optimization parameters which minimizes the memory footprint. The memory-efficient perturbation routine is shown in 5. The parameters are perturbed in groups rather than individually, i.e. in Algorithm 5, each θ_i denotes a parameter group (e.g. an entire weight matrix). The scaling factor $s \in \{1, -2\}$ is used to perturb the parameters in a forward and backward direction as required in central difference approximations.

Algorithm 5 Memory-Efficient Parameter Perturbation

Design choices: Scaling factor $s \in \{1, -2\}$, perturbation size μ
Input: Parameters $\boldsymbol{\theta}$, random seed r
Return: Updated parameters $\boldsymbol{\theta}$

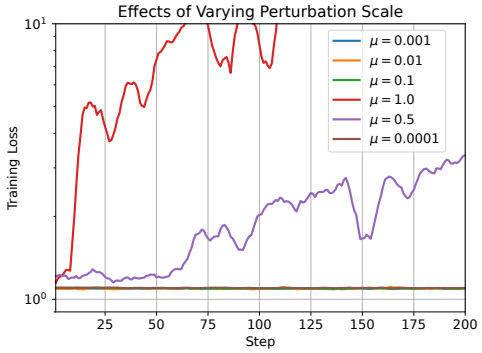
begin method
1. Set random seed r
for $\theta_i \in \boldsymbol{\theta}$ **do**
  2. $z_i \sim \mathcal{N}(0, 1)$
  3. $\theta_i \leftarrow \theta_i + s * z_i * \mu$
end for
end

In this work, experiments were conducted with single SPSA estimators which require exactly 2 forward passes. In p-SPSA, p estimators are computed and averaged. A total of $2p$ forward passes are required to compute a p-SPSA estimator. We used the default setting of $p = 1$ suggested in [34] for both MeZO and MeZO-SVRG implementations.

Role of the Perturbation Parameter

We investigated the role of the perturbation parameter μ in MeZO-SVRG. Recall that μ defines the forward and backward perturbation scale when computing SPSA estimators (3.5) and (3.6). We know from [35] that the SPSA estimator is asymptotically unbiased as $\mu \to 0$. We wanted to see the practical effects of different μ settings for MeZO-SVRG. Thus we carried out an ablation study where the perturbation parameter was varied. We fine-tune DistilBert [48] on the MNLI [46] dataset. The experiment settings are summarized in Figure 3.4b.

Figure 3.4a shows how the different values of μ affected the fine-tuning process of the MeZO-SVRG algorithm. We observe that for a sufficiently small values of μ (i.e. smaller than $1e-1$) we see no noticeable difference in performance, while larger μ result in diverging behaviour. Similar findings were also empirically corroborated in [34]. Thus, throughout our work we used the default value of $\mu = 1e-3$.



(a)

Algorithm	Hyperparameters	Values
MeZO-SVRG	Batch size	$\{64\}\times$
	Learning rate (η_1)	$\{1e-4\}\times$
	Learning rate (η_2)	$\{1e-6\}\times$
	μ	$\{1, 0.5, 1e-1, 1e-2, 1e-4\}\times$
	q	$\{2\}\times$
	Total Steps	$\{200\}$

(b)

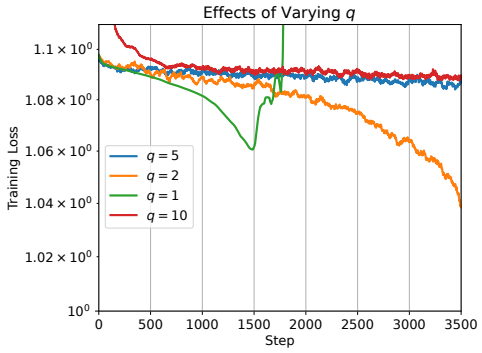
Figure 3.4: a) Shows the effects of varying the perturbation scale on the performance of MeZO-SVRG. b) Shows the hyperparameter settings used in this experiment.

Role of q

The parameter q plays a significant role in the performance of MeZO-SVRG (Algorithm 3). Concretely, q determines the frequency of fullbatch update steps in the algorithm: smaller q increases the regularity of fullbatch updates. We perform an ablation to better understand the extent to which fullbatch updates help or hinder the MeZO-SVRG performance. We consider the task of fine-tuning the DistilBert [48] model on the MNLI [46] dataset. The experiment setup is summarized in Figure 3.5b.

Figure 3.5a shows the training curves of MeZO-SVRG for different settings of q over 3500 steps. Increasing the frequency of fullbatch update steps enhances the convergence rate. However, our findings also indicate that a combination of fullbatch and minibatch updates (with $q \geq 2$) con-

tributes to a more stable algorithm performance compared to exclusively using fullbatch updates (when $q = 1$).



(a)

Algorithm	Hyperparameters	Values
MeZO-SVRG	Batch size	$\{64\}\times$
	Learning rate (η_1)	$\{1e-4\}\times$
	Learning rate (η_2)	$\{1e-6\}\times$
	μ	$\{1e-3\}\times$
	q	$\{1, 2, 5, 10\}\times$
	Total Steps	$\{3500\}$

(b)

Figure 3.5: a) Shows the effects of varying q on the convergence performance MeZO-SVRG. b) Shows the hyperparameter settings used in this experiment.

Improved Robustness to Batch Size

In Figures 3.1a, 3.1b and 3.1c we emphasize one of the practical drawbacks of MeZO with respect to instability with small batch sizes. We saw this behavior even in the more benign prompted setting. In Figure 3.6, we compare the behavior of MeZO-SVRG and MeZO when fine-tuning RoBERTa-large [42] on the SST-2 dataset in the prompt-free setting. The plot showcases MeZO-SVRG's advantage as a low-variance method with improved robustness to different batch sizes. In particular, MeZO's tendencies of diverging with smaller batch sizes are mitigated by MeZO-SVRG. Note that this improvement already becomes apparent over the first 100 iterations of fine-tuning.

Approximating Fullbatch Estimators with Large Batches

For sufficiently large training datasets, estimating the fullbatch gradient estimator is prohibitive and time-consuming. Thus we carry out an ablation study to see the effects on the MeZO-SVRG performance when approximating the fullbatch gradient estimator with a large-batch estimator. Specifically, we carry out partial-parameter fine-tuning of DistilBert on a training set of 512 samples for 8000 steps. We choose a mini-batch size of 64 which is consistent across experiment runs. This ablation study is carried out in the half-precision (BF16) setting. We approximate the fullbatch (512 samples) with large batch sizes of 256 and 128. The fine-tuning performances are summarized in Table 3.10. The obtained results are comparable, suggesting that the large batch-based gradient estimation offers a viable approximation of the fullbatch gradient estimator.

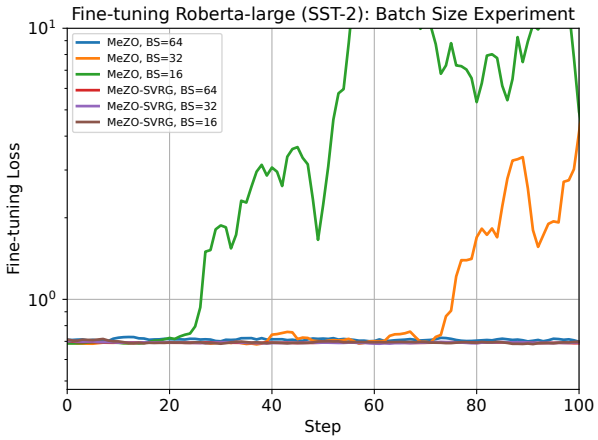


Figure 3.6: Shows improved robustness to smaller batch sizes for MeZO-SVRG compared to MeZO when fine-tuning RoBERTa-large on the SST-2 dataset.

Table 3.10: Performance of partial-parameter fine-tuning of DistilBert with half-precision when approximating the fullbatch with large batch sizes. Partial FT refers to partial-parameter fine-tuning (see Section 3.11 for details).

Task	**Method**	**Fine-tuning Loss** ↓	**Test Accuracy** (%)↑	**Queries** ($\times 10^3$) ↓
SST-2 (Full FT)	MeZO-SVRG (fullbatch= 512)	0.4393	70	2560
	MeZO-SVRG (Large batch= 256)	0.4946	71	1536
	MeZO-SVRG (Large batch= 128)	0.5502	69	1024

Learning Rate Scheduling

In our implementation, we couple the MeZO-SVRG method with a basic learning rate annealing schedule. This schedule is shown in Algorithm 6. This scheduling scheme operates on feedback from training loss values. We compute the average loss values in consecutive epochs. If an increasing trend of average losses is observed, the learning rates are annealed with a factor of α. Specifically, if the ratio of leading and trailing average losses is above threshold κ, we anneal the learning rates. In our experiments we set $\kappa = 1.05$ and annealing factor $\alpha = 5$.

Algorithm 6 Learning Rate Scheduling for MeZO-SVRG

Input: Learning rates η_1, η_2, annealing factor α, losses L, annealing threshold κ, total number of batches in an epoch w

begin method

1. $m_1 \leftarrow \texttt{mean}(L[-w, :])$
2. $m_2 \leftarrow \texttt{mean}(L[-2w, -w])$

if $\frac{m_1}{m_2} > \kappa$ **then**

3. $\eta_1 \leftarrow \frac{\eta_1}{\alpha}$, $\eta_2 \leftarrow \frac{\eta_2}{\alpha}$

end if

end

3.13 Additional Results: Fine-tuning DistilBert

Hyperparameter Selection

Table 3.11 shows the hyperparameter grid optimized over in the DistilBert [48] experiment. The hyperparameter search was done by running the different algorithms for 1K steps on the MNLI [46] dataset and selecting the best configuration. The chosen configuration was then used for a longer fine-tuning runs for all considered tasks, i.e. 200K steps for MeZO and 50K steps for MeZO-SVRG.

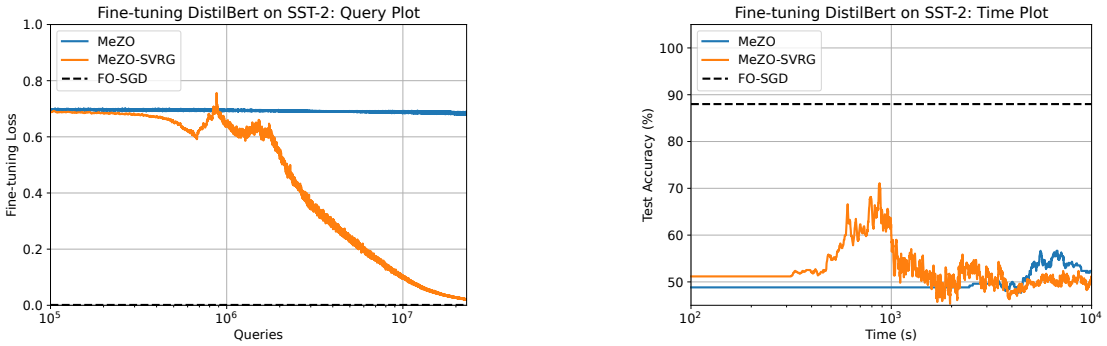
Table 3.11: The hyperparameter grid optimized over for the DistilBert [48] experiments. In the case of MeZO-SVRG we use the learning rate schedule proposed in Algorithm 6. The bold values indicate the configuration used to generate the final results.

Algorithm	**Hyperparameters**	**Values**
MeZO	Batch size	$\{32, \mathbf{64}\}\times$
	Learning rate	$\{1e{-}4, 1e{-}5, \mathbf{1e{-}6}\}\times$
	μ	$\{\mathbf{1e{-}3}\}\times$
	Total Steps	$\{\mathbf{200K}\}$
MeZO-SVRG	Batch size	$\{32, \mathbf{64}\}\times$
	Learning rate (η_1)	$\{\mathbf{1e{-}3}, 1e{-}4\}\times$
	Learning rate (η_2)	$\{1e{-}5, \mathbf{1e{-}6}\}\times$
	μ	$\{\mathbf{1e{-}3}\}\times$
	q	$\{\mathbf{2}, 5, 10\}\times$
	Total Steps	$\{\mathbf{50K}\}$
FO-SGD	Batch size	$\{32, \mathbf{64}\}\times$
	Learning rate	$\{1e{-}2, \mathbf{1e{-}3}, 1e{-}4\}\times$
	Total Steps	$\{\mathbf{1K}\}$

Convergence Performance

We fine-tune Distilbert [48] on the SST-2 [40] dataset. In Figure 3.7a, we show the improved convergence performance of MeZO-SVRG over MeZO. MeZO-SVRG is able to significantly reduce the convergence gap compared to the FO-SGD baseline. Figure 3.7b shows the evolution of the test accuracy over time. Observe that MeZO-SVRG achieves a significant improvement over MeZO in test performance. Moreover, MeZO-SVRG surpasses the peak test accuracy achieved by MeZO in over an order of magnitude less time.

Additional Results



(a) (b)

Figure 3.7: Performance of MeZO-SVRG and MeZO when fine-tuning Distilbert [48] on the SST-2 [40] dataset. The dashed line serves as a reference to the training loss/test accuracy achieved by FO-SGD. (a) MeZO-SVRG is able to significantly reduce the convergence gap to FO-SGD compared to MeZO. (b) MeZO-SVRG surpasses the peak test performance of MeZO in an order of magnitude less time.

3.14 Additional Results: Fine-tuning RoBERTa-large

Hyperparameter selection

Table 3.13 presents the hyperparameters searched over in our RoBERTa-large [42] experiment. The hyperparameter search was done by fine-tuning the model on the MNLI [46] dataset for 1K steps and selecting the best configuration. This selected configuration was subsequently applied to extended fine-tuning sessions across all considered tasks. For our final results, MeZO-SVRG was run for 24K steps and MeZO was run for 96K steps.

Additional Results

Table 3.12: Experiments on DistilBERT (with 512 fine-tuning examples). FO refers to first-order methods. Full FT refers to full-parameter fine-tuning and Partial FT refers to partial-parameter fine-tuning (see Section 3.11 for details).

Task	**Method**	**Fine-tuning Loss** ↓	**Test Accuracy** (%)↑	**Queries** ($\times 10^3$) ↓
MNLI (Full FT)	MeZO	1.0908	36	25600
	MeZO-SVRG	**0.0757**	**46**	25600
	FO-SGD	0.0101	59	64
MNLI (Partial FT)	MeZO	1.0925	35	25600
	MeZO-SVRG	**0.2775**	**47**	25600
	FO-SGD	0.2617	48	64
QNLI (Full FT)	MeZO	0.6914	50	25600
	MeZO-SVRG	**0.2335**	**68**	25600
	FO-SGD	0.0372	78	64
QNLI (Partial FT)	MeZO	0.6929	52	25600
	MeZO-SVRG	**0.2925**	**65**	25600
	FO-SGD	0.4176	59	64
SST-2 (Full FT)	MeZO	0.6822	52	25600
	MeZO-SVRG	**0.0203**	**72**	25600
	FO-SGD	0.0121	88	64
SST-2 (Partial FT)	MeZO	0.6990	51	25600
	MeZO-SVRG	**0.1034**	**74**	25600
	FO-SGD	0.0507	85	64
CoLA (Full FT)	MeZO	0.6408	62	25600
	MeZO-SVRG	**0.2807**	**68**	25600
	FO-SGD	0.0159	70	64
CoLA (Partial FT)	MeZO	0.6422	60	25600
	MeZO-SVRG	**0.3617**	**67**	25600
	FO-SGD	0.44719	66	64

3.15 Additional Results: Additional Results for fine-tuning Autoregressive Models

Hyperparameter Selection

Table 3.16 presents the hyperparameter grid searched over for the experiments on autoregressive models. The hyperparameter search was conducted by fine-tuning the models on the MNLI [46] dataset for 100 steps and selecting the best configuration. This selected configuration was used in extended fine-tuning sessions across all considered tasks. For our final results, MeZO-SVRG was

Table 3.13: The hyperparameter grid optimized over for the RoBERTa-large [42] experiments. In the case of ZO-SVRG we use the learning rate schedule proposed in Algorithm 6. The bold values indicate the configuration used to generate the final results.

Algorithm	**Hyperparameters**	**Values**
MeZO	Batch size	$\{32, \mathbf{64}\}\times$
	Learning rate	$\{1e{-}4, 1e{-}5, \mathbf{1e{-}6}\}\times$
	μ	$\{\mathbf{1e{-}3}\}\times$
	Total Steps	$\{\mathbf{96K}\}$
MeZO-SVRG	Batch size	$\{32, \mathbf{64}\}\times$
	Learning rate (η_1)	$\{1e{-}4, 5e{-}5, \mathbf{1e{-}5}\}\times$
	Learning rate (η_2)	$\{1e{-}5, \mathbf{1e{-}6}\}\times$
	μ	$\{\mathbf{1e{-}3}\}\times$
	q	$\{\mathbf{2}, 5, 10\}\times$
	Total Steps	$\{\mathbf{24K}\}$
FO-SGD	Batch size	$\{32, \mathbf{64}\}\times$
	Learning rate	$\{1e{-}3, \mathbf{1e{-}4}, 1e{-}5\}\times$
	Total Steps	$\{\mathbf{1K}\}$

run for 8K steps and MeZO was run for 32K steps.

Convergence Performance

We fine-tune GPT2 [49] and OPT-2.7B [50] on the QNLI [37] dataset. In Figures 3.8a and 3.9a, we show the improved convergence performance of MeZO-SVRG over MeZO. For both models, MeZO-SVRG is able to significantly reduce the convergence gap compared to the FO-SGD baseline. Figures 3.8b and 3.9b show the evolution of the test accuracy over time. As with the experiments on masked models, MeZO-SVRG achieves a significant improvement over MeZO in test performance.

Additional Results

Tables 3.17 and 3.18 present extended results on the fine-tuning tasks for GPT2 [49] and OPT-2.7B [50].

Table 3.14: Experiments on RoBERTa-large (with 512 fine-tuning examples). Here partial refers to fine-tuning the last layers of the model (see Section 3.11 for details). FO refers to first-order methods. Full FT refers to full-parameter fine-tuning and Partial FT refers to partial-parameter fine-tuning.

Task	**Method**	**Fine-tuning Loss** ↓	**Test Accuracy** (%)↑	**Queries** ($\times 10^3$) ↓
MNLI (Full FT)	MeZO	0.9447	43	12288
	MeZO-SVRG	**0.8125**	**49**	12288
	FO-SGD	0.0292	85	64
MNLI (Partial FT)	MeZO	1.0729	42	12288
	MeZO-SVRG	**0.8176**	**43**	12288
	FO-SGD	0.9859	52	64
QNLI (Full FT)	MeZO	0.5845	59	12288
	MeZO-SVRG	**0.2750**	**80**	12288
	FO-SGD	0.01426	89	64
QNLI (Partial FT)	MeZO	0.6885	50	12288
	MeZO-SVRG	**0.4557**	**67**	12288
	FO-SGD	0.5974	72	64
SST-2 (Full FT)	MeZO	0.69155	56	12288
	MeZO-SVRG	**0.1336**	**84**	12288
	FO-SGD	0.1086	96	64
SST-2 (Partial FT)	MeZO	0.6837	54	12288
	MeZO-SVRG	**0.5896**	**72**	12288
	FO-SGD	0.5786	89	64
CoLA (Full FT)	MeZO	0.5062	68	12288
	MeZO-SVRG	**0.0644**	**79**	12288
	FO-SGD	0.0099	85	64
CoLA (Partial FT)	MeZO	0.5868	65	12288
	MeZO-SVRG	**0.3538**	**79**	12288
	FO-SGD	0.4075	84	64

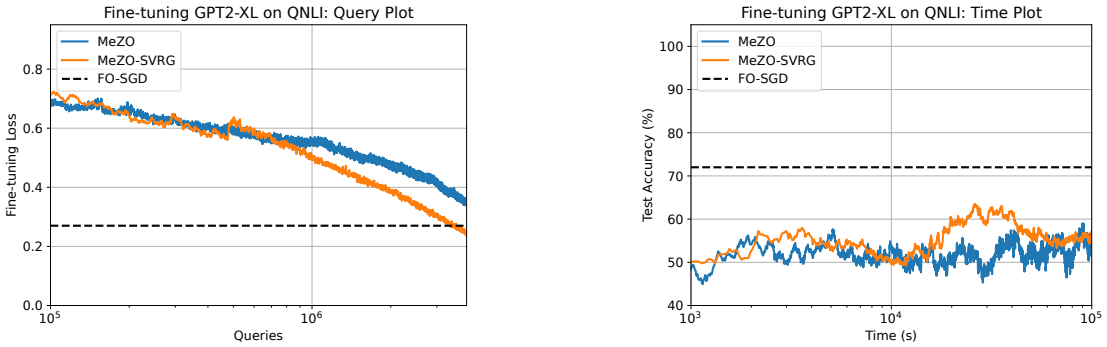
3.16 Additional Results: Memory Usage and Computation Time

Memory Profiling

We performed memory profiling experiments without any advanced memory-saving options such as lowering bit precision [60] or gradient check-pointing [62]. We used full (f32) floating-point precision.

Table 3.15: Experiments on RoBERTa-large (with 512 fine-tuning examples) in the prompted setting. Here partial refers to fine-tuning the last layers of the model (see Section 3.11 for details). FO refers to first-order methods. Full FT refers to full-parameter fine-tuning and Partial FT refers to partial-parameter fine-tuning.

Task	**Method**	**Fine-tuning Loss** ↓	**Test Accuracy** (%)↑	**Queries** ($\times 10^3$) ↓
MNLI with Prompt (Full FT)	MeZO	0.0076	73	12288
	MeZO-SVRG	**0.0058**	**75**	12288
	FO-SGD	0.0036	96	64
MNLI with Prompt (Partial FT)	MeZO	0.4614	65	12288
	MeZO-SVRG	**0.3177**	65	12288
	FO-SGD	0.3676	81	64
SST-2 With Prompt (Full FT)	MeZO	**0.2959**	**93**	12288
	MeZO-SVRG	0.3063	92	12288
	FO-SGD	0.1578	93	64
SST-2 with Prompt (Partial FT)	MeZO	**0.3280**	89	12288
	MeZO-SVRG	0.3393	89	12288
	FO-SGD	0.2981	90	64



(a) (b)

Figure 3.8: Convergence performance of MeZO-SVRG, MeZO and FO-SGD when fine-tuning GPT2 [49] on the QNLI [37] dataset. The dashed line serves as a reference to the training loss achieved by FO-SGD. MeZO-SVRG is able to surpass the fine-tuning loss obtained by FO-SGD. It also improves on the test accuracy attained by MeZO.

In the first experiment, we measured the memory requirement needed to run the different methods on full-parameter fine-tuning tasks. The MNLI [46] dataset was used to fine-tune autoregressive models GPT2 [49], OPT-2.7B, OPT-6.7B [50]. We set the input sequence length to the maximum context length for each model, i.e. 1024 for GPT2 and 2048 for the OPT models. The batch size was set to 1. Figure 3.3 shows the peak memory consumption in GB as reported

Table 3.16: The hyperparameter grid optimized over for the GPT2 [49] and OPT-2.7B [50] experiments. In the case of MeZO-SVRG we use the learning rate schedule proposed in Algorithm 6. The bold values indicate the configuration used to generate the final results for both models.

Algorithm	**Hyperparameters**	**Values**
MeZO	Batch size	$\{32, \mathbf{64}\}\times$
	Learning rate	$\{1e{-}6, \mathbf{5e{-}6}, 1e{-}7\}\times$
	μ	$\{\mathbf{1e{-}3}\}\times$
	Total Steps	$\{\mathbf{32K}\}$
MeZO-SVRG	Batch size	$\{32, \mathbf{64}\}\times$
	Learning rate (η_1)	$\{1e{-}4, \mathbf{5e{-}5}, 1e{-}5\}\times$
	Learning rate (η_2)	$\{\mathbf{1e{-}6}\}\times$
	μ	$\{\mathbf{1e{-}3}\}\times$
	q	$\{\mathbf{2}, 5, 10\}\times$
	Total Steps	$\{\mathbf{8K}\}$
FO-SGD	Batch size	$\{8, \mathbf{16}\}\times$
	Learning rate	$\{\mathbf{1e{-}4}, 1e{-}5\}\times$
	Total Steps	$\{\mathbf{500}\}$

by the `nvidia-smi` command. The peak memory consumption was obtained after executing the methods for at least 100 steps. Table 3.3 presents the largest GPT/OPT model that can be fit for each method under the aforementioned settings on single Nvidia A100 40GB and H100 80GB GPUs.

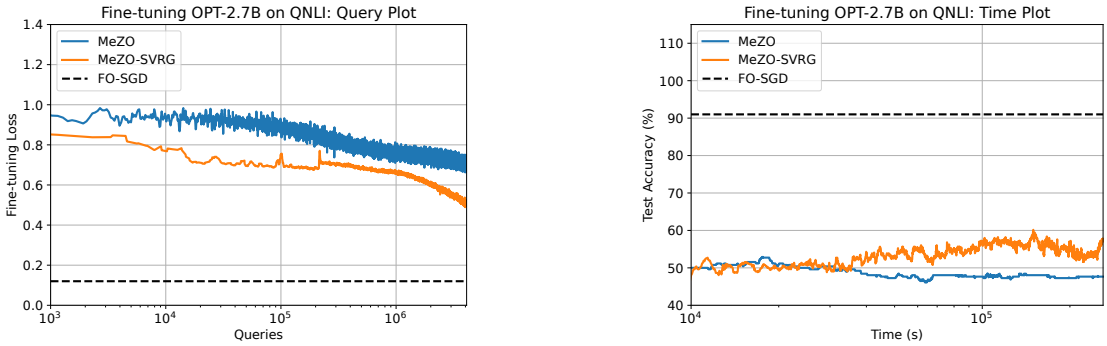
In the second experiment, we measured how the memory usage for the different methods scales with increasing batch size. We fine-tuned RoBERTa-large [42] on the MNLI [46] dataset. The input sequence length was set to a constant 128 and we varied the batch size $\{16, 32, 64\}$. The memory consumption was again measured using the `nvidia-smi` command and measurements were taken after running the methods for at least 100 steps. Table 3.3 summarizes the results.

We finally also measured how the memory usage varies for the considered algorithms when using a fixed batch size (64) and changing the context length of the input. We used a similar setting to the second experiment: fine-tuning RoBERTa-large [42] on the MNLI [46] dataset. The input context length was varied $\{128, 256, 512\}$ and the memory consumption was measured using the `nvidia-smi` command. Table 3.3 reports the results.

We replicated all experiments in the half-precision (BF16) setting; the results are given in Table 3.25.

Computation Time

We compared the speed of MeZO-SVRG and MeZO [34] by measuring the time taken by each method to achieve the test performance attained by MeZO. These measurements are based on



(a) (b)

Figure 3.9: Performance of MeZO-SVRG, MeZO and FO-SGD when fine-tuning OPT-2.7B [50] on the QNLI [37] dataset. The dashed line serves as a reference to the training loss/test accuracy achieved by FO-SGD. MeZO-SVRG is able to reduce the convergence gap to FO-SGD compared to MeZO and improve on the test accuracy.

Table 3.17: Experiments on GPT2 (with 512 fine-tuning examples). FO refers to first-order methods. This table summarizes results for full-parameter fine-tuning.

Task	**Method**	**Fine-tuning Loss** ↓	**Test Accuracy** (%)↑	**Queries** ($\times 10^3$) ↓
MNLI (Full FT)	MeZO	0.6526	41	4096
	MeZO-SVRG	**0.4116**	**53**	4096
	FO-SGD	0.5924	69	8
QNLI (Full FT)	MeZO	0.3351	58	4096
	MeZO-SVRG	**0.2372**	**63**	4096
	FO-SGD	0.2799	72	8
SST-2 (Full FT)	MeZO	0.3240	59	4096
	MeZO-SVRG	**0.2024**	**65**	4096
	FO-SGD	0.2343	72	8
CoLA (Full FT)	MeZO	0.3544	68	4096
	MeZO-SVRG	**0.2455**	**69**	4096
	FO-SGD	0.3855	78	8

fine-tuning GPT2 [49] and OPT-2.7B [50] on all considered datasets. Table 3.4 summarizes the results.

Table 3.18: Experiments on OPT-2.7B (with 512 fine-tuning examples). FO refers to first-order methods. This table summarizes results for full-parameter fine-tuning.

Task	**Method**	**Fine-tuning Loss** $\downarrow$	**Test Accuracy** (%)$\uparrow$	**Queries** ($\times 10^3$) $\downarrow$
MNLI (Full FT)	MeZO	1.0875	42	4096
	MeZO-SVRG	**0.8159**	**52**	4096
	FO-SGD	0.3305	78	8
QNLI (Full FT)	MeZO	0.7026	53	4096
	MeZO-SVRG	**0.4634**	**60**	4096
	FO-SGD	0.1222	91	8
SST-2 (Full FT)	MeZO	0.6530	61	4096
	MeZO-SVRG	**0.5501**	**65**	4096
	FO-SGD	0.0167	98	8
CoLA (Full FT)	MeZO	0.5823	62	4096
	MeZO-SVRG	**0.5335**	**67**	4096
	FO-SGD	0.1724	94	8

3.17 Additional Results: Half-Precision Experiments

In the section, we run preliminary experiments to evaluate the considered fine-tuning algorithms on the half-precision (BF16) setting.

Half-Precision Experiments on DistilBert

The hyperparameter grid that was optimized over for the DistilBert experiments in the half-precision setting is presented in Table 3.19. As each iteration under the half-precision setting is faster than under the full-precision setting, we run experiments for longer. Specifically, we run MeZO-SVRG for 80K steps, MeZO for 400K steps and FO-SGD for 2K steps. The results are summarized in Table 3.20.

Half-Precision Experiments on RoBERTa-large

The hyperparameter grid that was optimized over for the DistilBert experiments in the half-precision setting is presented in Table 3.21. As each iteration under the half-precision setting is faster than under the full-precision setting, we run experiments for longer. Specifically, we run MeZO-SVRG for 40K steps, MeZO for 200K steps and FO-SGD for 1K steps. The results are summarized in Table 3.22.

Table 3.19: The hyperparameter grid optimized over for the half-precision DistilBert [48] experiments. In the case of MeZO-SVRG we use the learning rate schedule proposed in Algorithm 6. The bold values indicate the configuration used to generate the final results.

Algorithm	**Hyperparameters**	**Values**
MeZO	Batch size	$\{32, \mathbf{64}\}\times$
	Learning rate	$\{1e{-}4, \mathbf{1e{-}5}, 1e{-}6\}\times$
	μ	$\{\mathbf{1e{-}2}\}\times$
	Total Steps	$\{\mathbf{400K}\}$
MeZO-SVRG	Batch size	$\{32, \mathbf{64}\}\times$
	Learning rate (η_1)	$\{\mathbf{1e{-}3}, 1e{-}4\}\times$
	Learning rate (η_2)	$\{\mathbf{1e{-}5}, 1e{-}6\}\times$
	μ	$\{\mathbf{1e{-}2}\}\times$
	q	$\{\mathbf{2}, 5\}\times$
	Total Steps	$\{\mathbf{80K}\}$
FO-SGD	Batch size	$\{32, \mathbf{64}\}\times$
	Learning rate	$\{\mathbf{1e{-}2}, 1e{-}3, 1e{-}4\}\times$
	Total Steps	$\{\mathbf{2K}\}$

Table 3.20: Half-precision experiments on DistilBERT (with 512 fine-tuning examples). FO refers to first-order methods. Partial FT refers to partial-parameter fine-tuning (see Section 3.11 for details).

Task	**Method**	**Fine-tuning Loss** $\downarrow$	**Test Accuracy** (%)$\uparrow$	**Queries** ($\times 10^3$) $\downarrow$
MNLI (Partial FT)	MeZO	1.0892	43	51200
	MeZO-SVRG	**0.8746**	**45**	51200
	FO-SGD	0.3508	51	128
QNLI (Partial FT)	MeZO	0.6904	60	51200
	MeZO-SVRG	**0.5416**	**64**	51200
	FO-SGD	0.2998	66	128
SST-2 (Partial FT)	MeZO	0.6889	61	51200
	MeZO-SVRG	**0.3887**	**79**	51200
	FO-SGD	0.0555	82	128
CoLA (Partial FT)	MeZO	0.6420	66	51200
	MeZO-SVRG	**0.6170**	**71**	51200
	FO-SGD	0.4218	70	128

Half-Precision Experiments on OPT-6.7B

The hyperparameter grid optimized for the OPT-6.7B experiments in the half-precision setting is detailed in Table 3.23. We conducted the MeZO-SVRG experiments for 8k steps, MeZO for 24k

Table 3.21: The hyperparameter grid optimized over for the half-precision RoBERTa-large [42] experiments. In the case of MeZO-SVRG we use the learning rate schedule proposed in Algorithm 6. The bold values indicate the configuration used to generate the final results.

Algorithm	**Hyperparameters**	**Values**
MeZO	Batch size	$\{\mathbf{64}\}\times$
	Learning rate	$\{1e{-}4, \mathbf{1e{-}5}, 1e{-}6\}\times$
	μ	$\{\mathbf{1e{-}3}\}\times$
	Total Steps	$\{\mathbf{200K}\}$
MeZO-SVRG	Batch size	$\{\mathbf{64}\}\times$
	Learning rate (η_1)	$\{\mathbf{1e{-}4}, 1e{-}5\}\times$
	Learning rate (η_2)	$\{\mathbf{1e{-}5}, 1e{-}6\}\times$
	μ	$\{\mathbf{1e{-}3}\}\times$
	q	$\{\mathbf{2}, 5\}\times$
	Total Steps	$\{\mathbf{40K}\}$
FO-SGD	Batch size	$\{\mathbf{64}\}\times$
	Learning rate	$\{\mathbf{1e{-}2}, 1e{-}3, 1e{-}4\}\times$
	Total Steps	$\{\mathbf{1K}\}$

Table 3.22: Half-precision experiments on RoBERTa-large (with 512 fine-tuning examples). FO refers to first-order methods. Partial FT refers to partial-parameter fine-tuning (see Section 3.11 for details).

Task	**Method**	**Fine-tuning Loss** $\downarrow$	**Test Accuracy** (%)$\uparrow$	**Queries** ($\times 10^3$) $\downarrow$
MNLI (Partial FT)	MeZO	1.0898	42	25600
	MeZO-SVRG	**1.0695**	**43**	25600
	FO-SGD	0.1820	55	64
QNLI (Partial FT)	MeZO	0.6835	62	25600
	MeZO-SVRG	**0.6070**	**68**	25600
	FO-SGD	0.3112	67	64
SST-2 (Partial FT)	MeZO	0.6630	66	25600
	MeZO-SVRG	**0.5278**	**77**	25600
	FO-SGD	0.1356	93	64
CoLA (Partial FT)	MeZO	0.6308	66	25600
	MeZO-SVRG	**0.5781**	**69**	25600
	FO-SGD	0.1537	88	64

steps, and FO-SGD for 1k steps. The outcomes of these experiments are summarized in Table 3.24. We include the BoolQ dataset from the SuperGLUE [38] benchmark to evaluate a more

challenging fine-tuning task.

Table 3.23: The hyperparameter grid optimized over for the half-precision OPT-6.7B [50] experiments. In the case of MeZO-SVRG we use the learning rate schedule proposed in Algorithm 6. The bold values indicate the configuration used to generate the final results.

Algorithm	**Hyperparameters**	**Values**
MeZO	Batch size	$\{\mathbf{128}\}\times$
	Learning rate	$\{1e{-}5, \mathbf{1e{-}6}\}\times$
	μ	$\{\mathbf{1e{-}3}\}\times$
	Total Steps	$\{\mathbf{24K}\}$
MeZO-SVRG	Batch size	$\{\mathbf{128}\}\times$
	Learning rate (η_1)	$\{\mathbf{1e{-}4}, 1e{-}5\}\times$
	Learning rate (η_2)	$\{1e{-}5, \mathbf{1e{-}6}\}\times$
	μ	$\{\mathbf{1e{-}3}\}\times$
	q	$\{\mathbf{2}, 5\}\times$
	Total Steps	$\{\mathbf{8K}\}$
FO-SGD	Batch size	$\{\mathbf{64}\}\times$
	Learning rate	$\{1e{-}3, \mathbf{1e{-}4}\}\times$
	Total Steps	$\{\mathbf{1K}\}$

Table 3.24: Half-precision experiments on OPT-6.7B (with 512 fine-tuning examples). FO refers to first-order methods. Full FT refers to full-parameter fine-tuning (see Section 3.11 for details).

Task	**Method**	**Fine-tuning Loss** $\downarrow$	**Test Accuracy** (%)$\uparrow$	**Queries** ($\times 10^3$) $\downarrow$
SST-2 (Full FT)	MeZO	0.5318	74	6144
	MeZO-SVRG	**0.5278**	**77**	6144
	FO-SGD	0.103	91	128
BoolQ (Full FT)	MeZO	0.6259	65	6144
	MeZO-SVRG	**0.5703**	**69**	6144
	FO-SGD	0.2872	84	128

Memory Profiling with Half-Precision

	Largest OPT/GPT that can fit	**Memory Usage in GB for RoBERTa-large**				
		Fixed context length (cl=128)			Fixed batch size (bs=64)	
Method	A100 (40GB)	bs $= 16$	bs $= 32$	bs $= 64$	cl $= 256$	cl $= 512$
MeZO	13B	1.03	1.13	1.25	1.39	2.66
MeZO-SVRG	**6.7B**	**2.10 (39%)**	**2.11 (66%)**	**2.12 (79%)**	**2.27 (90%)**	**3.66**
FO-SGD	2.7B	3.42	5.81	9.83	21.87	OOM
FO-Adam	1.3B	5.85	8.07	12.16	24.29	OOM

Table 3.25: Memory profiling with half-precision. Shows the largest AR models that can fit on single 40 GPUs. We also measure the memory usage under different batch sizes (bs) and context lengths (cl) when fine-tuning RoBERTa-large. Percentages indicate the memory savings with respect to FO-SGD.

Part II

Enhancing Training with Differentiable Optimization

Chapter 4

Meta-Learning Parameterized First-Order Optimizers

4.1 Introduction

First-order optimization methods underpin a wide range of modern control and machine learning (ML) techniques. The field of deep learning, including domains such as computer vision [11], [67], natural language processing [68], deep reinforcement learning [69], and robotics [70], has yielded revolutionary results when trained with variants of gradient descent such as stochastic gradient descent (SGD) [31] and Adam [32]. Algorithms like projected and conditional gradient descent extend the class of first-order methods to accommodate problems with constraints such as matrix completion, or training well-posed implicit deep models [3], [71].

While this proliferation of methods has facilitated rapid advances across the control and ML communities, designing update rules tailored to specific problems still remains a challenge. This challenge is exacerbated by the fact that different domains are tasked with solving distinct problem types. The deep learning community is, for instance, tasked with solving high-dimensional non-convex problems whereas the optimal control community often deals with constrained convex problems where the constraints encode restrictions on the state space and system dynamics. Moreover, even different problem instances within a particular problem class may require significantly varying update rules. As an example, within deep learning, effective hyperparameter (e.g. learning rate) selection for algorithms such as Adam and SGD is highly dependent on the underlying model that is to be trained.

Contributions

This chapter proposes a new data-driven approach for optimization algorithm design based on differentiable convex optimization (DCO). This approach enables the use of previous optimization experience to propose update rules that efficiently solve new optimization tasks sampled from the same underlying problem class. We start by introducing the notion of DCO as a means to parameterize optimizers within the meta-learning framework. We then propose an efficient instantiation

of meta-training that can be leveraged by the DCO optimizer to learn appropriate meta-parameters. To illustrate the generality of the DCO meta-learning framework, we then formulate concrete differentiable quadratic optimizations to solve unconstrained optimization problems, namely, DCO Gradient (DCOG), DCO Momentum (DCOM) and DCO General Descent (DCOGD). These DCO instantiations are generalizations of existing first-order update rules, which in turn demonstrates that existing methods can be thought of special cases of the DCO meta-learning framework.

DCO also provides sufficient structure conducive to rigorous theoretical analysis for the meta-learning problem. We establish convergence guarantees for the DCOGD optimizer to the optimal first-order update rule that leads to one step convergence when considering a family of linear least squares problems. Finally, we illustrate the potential of our proposed DCO optimizer instantiations by comparing convergence speed with popular existing first-order methods on illustrative regression and system identification tasks.

Related Works

Meta-Learning

Deep learning has demonstrated exceptional performance in situations where there is an abundance of training data and computational resources [1], [67], [68], [72]. This, however, excludes many important applications where there is an inherent lack of data or where computation is very expensive. Meta-learning attempts to address this issue by gaining learning process experience on similarly structured tasks [73]. This learning-to-learn paradigm is aligned with the human and animal learning process which tends to improve with greater experience. Moreover, by making the learning process more efficient meta-learning targets the aforementioned issues of data and compute scarcity.

Meta-learning methods can be categorized into three broad classes. In [74], authors introduce a unifying framework that encapsulates a wide class of existing approaches.

Optimizer-focused methods aim to improve the underlying optimizer in the inner loop used to solve the tasks at hand by meta-learning optimizer initialization or hyperparameters. Within few-shot learning, Model Agnostic Meta Learning (MAML) and its variants use prior learning experience to meta-learn a model/policy initialization that requires just a few inner gradient steps to adapt to a new task [75], [76]. Other works aim to meta-learn optimizer hyperparameters. In [77], [78], authors attempt to identify optimal learning rate scheduling strategies. Another strategy within this category is to directly learn a parameterization of the optimizer. Due to the sequential structure of inner loop parameter updates, recurrent architectures have been considered in this space [79], [80]. The inner loop optimization has also been viewed as a sequential decision-making problem and consequently optimizers have also been characterized as policies within an RL setting [81].

Black-box methods represent the inner loop via a forward-pass of a single model. The learning process of the inner loop is captured by the activation layers of the underlying model. The inner loop learning can be instantiated as RNNs [82], [83], convolutional neural networks (CNN) [84] or

hyper-networks [85]. The meta-learning loop finds the hyperparameters of the inner loop network yielding good performance.

In non-parametric methods, the inner loop aims to identify feature extractors that enable the matching of validation and training samples to yield an accurate prediction using the matched training label. The meta-loop aims to identify the class of feature extractors that transform the data samples into an appropriate space where matching is viable [86], [87].

Implicit Layers

Recent work has proposed a novel viewpoint wherein deep learning can be instantiated using implicit prediction rules rather than as conventional explicit feedforward architectures [3], [13], [14]. In [3] and [13] authors formalize how deep equilibrium models, characterized by nonlinear fixed point equations, represent weight-tied, infinite-depth networks. In this framework, [3] demonstrates how the aforementioned models are able to generalize most of the popular deep learning architectures. In [14], authors propose neural ordinary differential equations (ODE): an alternative instantiation of an implicit layer where the layer output is the solution to an ODE. This is shown to be an expressive model class yielding particularly impressive results when processing sequential data. Implicit layers have also been characterized as differentiable optimization layers. The work [88] introduces differentiable quadratic optimization (QP) layers that can be incorporated within deep learning architectures. In [89] authors develop software to differentiate through defined convex optimization problems. Some notable applications of differentiable optimization layers include parameterizing model predictive control policies [90] and representing a maximum satisfiability (MAXSAT) solver [91].

Notation

Throughout this chapter, we consider the problem of solving a task $\mathcal{T}$ which consists of an optimization problem and an evaluation step. The optimization problems are characterized with a loss function $l(\theta)$ over decision variables θ belonging to some parameter space $\Theta \subseteq \mathbb{R}^p$. For evaluation, we denote a validation criterion l^{val} that assesses the optimizer $\theta^\star$ found in the associated problem. We refer to solving the optimization over θ as the *inner loop* problem. At the meta-level we consider an algorithm $\textbf{Opt}(\,\cdot\,;\phi) : \Theta \to \Theta$ with meta-parameters ϕ which generates a sequence of parameter updates using first-order information to solve the inner loop problem. We denote the horizon of the parameter update sequence by T. By meta-training, we refer to the optimization over the meta-parameters over a training set of N tasks $\{\mathcal{T}_i\}_{i=1}^N$. For a vector-valued function $f(x) : \mathbb{R}^d \to \mathbb{R}^p$ we let the operator $\nabla_x f(\cdot) : \mathbb{R}^d \to \mathbb{R}^{d\times p}$ denote the gradient. If $f : \mathbb{R}^d \to \mathbb{R}$ is a scalar function, the Hessian of f is denoted by $\nabla_x^2 f(\cdot) : \mathbb{R}^d \to \mathbb{R}^{d\times d}$. We denote that a square symmetric matrix A is positive definite (all eigenvalues strictly positive) by $A \succ 0$. The vectorization of a matrix $A \in \mathbb{R}^{m\times n}$ is denoted by $\text{vec}(A) \in \mathbb{R}^{mn}$ and is constructed by stacking the columns of A. The Kronecker product of two matrices A and B is denoted $A \otimes B$. For a vector $x \in \mathbb{R}^n$ and $p \geq 1$, $\|x\|_p$ denotes the ℓ_p-norm of x. For $m \in \mathbb{N}_+$, we define $[m]$ to be the set $\{a \in \mathbb{N}_+ \mid a \leq m\}$, where $\mathbb{N}_+$ is the set of positive integer numbers. We define the operator $\odot$ as an elementwise mul-

tiplication. $\mathcal{U}(a, b)$ denotes the uniform probability distribution with support $[a, b]$ and $\mathcal{N}(\mu, \sigma^2)$ represents a univariate normal distribution centered at μ with standard deviation σ. Finally, we define $E_{\mathcal{D}}[\cdot]$ as the expectation operator over distribution $\mathcal{D}$.

4.2 Background

This section contextualizes our proposed framework. Section 4.2 illustrates how conventional first-order update rules can be typically expressed as the solution to a convex optimization problem. Section 4.2 then elaborates on the differentiable convex optimization methods that can be used to differentiate through the aforementioned inner loop gradient steps to update meta-parameters.

First-order methods

We consider a generic unconstrained optimization problem

$$\min_x f(x) \tag{4.1}$$

with differentiable objective f. First-order methods are a popular means to solve optimization problems of the form (4.1). The first-order property refers to the underlying methods' use of gradient information to generate a sequence of parameter iterates. Next we briefly survey a subset of important first-order methods that solve optimization problems of the form (4.1). We highlight how the update rules of these algorithms can be formulated as convex optimization problems themselves. This motivates the formulation of a generic parameterized convex optimizer to yield optimal parameter updates.

Gradient descent

Gradient descent (GD) is a standard first-order method used to solve a variety of unconstrained optimization problems. For an unconstrained optimization problem, GD updates aim to reconcile the notion of minimizing a linear approximation of the objective while simultaneously maintaining proximity to the current parameter iterate. This can be cast as the convex optimization

$$x^{(t+1)} = \arg\min_x \{\nabla f(x^{(t)})^\top (x - x^{(t)}) + \frac{\lambda}{2} ||x - x^{(t)}||_2^2\} \tag{4.2}$$

where $\lambda > 0$ is the step size. Solving (4.2) in closed-form yields the well-known GD update.

Gradient descent with Momentum

A popular practical variation of GD is to utilize the historical first-order information within the parameter update rule. This is referred to as GD with momentum. The contribution of historic

first-order information is captured by the notion of a state. More concretely, we define state update for $t > 1$ as

$$S^{(t+1)} = \beta S^{(t)} + (1 - \beta)\nabla f(x^{(t)}), \tag{4.3}$$

where $\beta \in [0, 1]$ is an averaging parameter and we initialize $S^{(1)} := \nabla f(x^{(1)})$. The convex update rule in this method substitutes $\nabla f(x^{(t)})$ with $S^{(t+1)}$:

$$x^{(t+1)} = \arg\min_x \{(S^{(t+1)})^\top (x - x^{(t)}) + \frac{\lambda}{2}||x - x^{(t)}||_2^2\} \tag{4.4}$$

Other notable first-order methods whose updates are defined via convex optimization problems are the proximal gradient (PG) [92], [93] and mirror descent (MD) [94], [95] methods. The former addresses unconstrained nondifferentiable problems whose objective is a composite function that can be decomposed into the sum of a differentiable and nondifferentiable part. The latter targets potentially constrained problems with updates that simultaneously minimize a linear approximation of the objective and a proximity term between parameter updates.

Differentiable Optimization Layers

We now present the formulation for a general DCO [89]:

$$\begin{aligned} D(x; \phi) := \arg\min_{y \in \mathbb{R}^n} \quad & f_0(x, y; \phi) \\ \text{s.t.} \quad & f_i(x, y; \phi) \leq 0 \quad \text{for } i \in [q], \\ & g_j(x, y; \phi) = 0 \quad \text{for } j \in [r], \end{aligned} \tag{4.5}$$

where $x \in \mathbb{R}^d$ is the optimization input and $y \in \mathbb{R}^n$ is the solution. Here optimization parameters are defined by a vector ϕ. The functions f_i parameterize inequality constraint functions which are convex in y and g_j parameterize affine equality constraints. As with the constraint functions, the objective f_0 is convex in the optimization variable y.

Note that this formulation defines a general parameterized convex optimization problem in the output y. The solution to the optimization is a function of the input x.

When embedding DCO as a layer within the deep learning context, we require the ability to differentiate through D with respect to ϕ when performing backpropagation. This is achieved via implicit differentiation through the Karush-Kuhn-Tucker (KKT) optimality conditions as proposed in [88], [96]. Particular instantiations of DCO, such as parameterized QPs, can enable more efficient backpropagation of gradients [88].

4.3 Meta-Optimization Framework

Consider the setting where we have N *training* tasks $\mathcal{T}_i = (l_i, l_i^{\text{val}})$ for $i \in [N]$, where each task consists of a tuple containing a training loss function l_i and an associated performance metric l_i^{val}.

Each of these tasks is sampled from an underlying distribution $\mathcal{D}$, i.e $\mathcal{T}_i \sim \mathcal{D}\ \forall i \in [N]$. For task $\mathcal{T}_i$, we consider the optimization

$$\min_{\theta_i \in \Theta} l_i(\theta_i) \tag{4.6}$$

where we aim to minimize loss l_i over the decision variable θ constrained to the set $\Theta \subset \mathbb{R}^p$. We let ϕ denote the set of meta-parameters that configure the method used to solve optimization (4.6), e.g. ϕ could include the learning rate in a gradient-based algorithm. The validation loss l_i^{val} is used to evaluate the final $\theta_i^\star$ recovered from solving (4.6). As motivation for this setup, we consider the general training-validation procedure seen in ML. Here l_i can be seen as the loss on training data with respect to model parameters θ and l_i^{val} denotes the loss on validation data. Note that for problems where the metric of interest is in fact the objective of (4.6), we can trivially define $l_i^{\text{val}} := l_i$.

In this meta-learning framework, the goal is to perform well on a new task $\mathcal{T}_{\text{target}} = (l_{\text{target}}, l_{\text{target}}^{\text{val}}) \sim \mathcal{D}$ using previous experience from tasks $\{\mathcal{T}_i\}_{i=1}^N$. Since $\mathcal{T}_{\text{target}}$ is sampled from the same distribution $\mathcal{D}$ as the training tasks, it has structural similarities that can be exploited by meta-learning.

Inner Optimization Loop

Depending on the structure of (4.6), several iterative methods exist to solve the considered problem. The chosen algorithm has an update rule that yields a sequence of parameter updates $\{\theta_i^{(t)}\}_{t=1}^T$ where T is defines the total number of updates and i indexes the associated task $\mathcal{T}_i$. Within the class of first-order methods, these update rules require computing or estimating (e.g. within reinforcement learning) the gradient $G_i^{(t)} := \nabla_\theta l_i(\theta_i^{(t)})$ to solve the inner optimization of task $\mathcal{T}_i$. The algorithm **Opt** applies the computed first-order and zeroth-order information at time step t along with an abstraction of past information encapsulated by state $S^{(t)}$ to yield both an updated parameter $\theta_i^{(t+1)}$ and state $S^{(t+1)}$:

$$(\theta_i^{(t+1)}, S_i^{(t+1)}) := \textbf{Opt}(\theta_i^{(t)}, S_i^{(t)}, G_i^{(t)}; \phi). \tag{4.7}$$

Here we characterize the optimizer with meta-parameters ϕ. Solving the inner loop problem to completion involves recursively applying (4.7) T times from an initial condition $\theta_i^{(1)}$ and history state $S_i^{(1)}$, which we denote by $\textbf{Opt}_T(\theta_i^{(1)}, S_i^{(1)}; \phi)$. Note that moving forward, unless made explicit, we suppress the return argument of the next state, i.e. we utilize the shorthand $\theta_i^{(t+1)} := \textbf{Opt}(\theta_i^{(t)}, S_i^{(t)}, G_i^{(t)}; \phi)$.

Meta-Learning Loop

The meta-learning loop wraps around the inner loop. It aims to find optimal meta-parameters $\phi^\star$ that ensure that for each task $\mathcal{T}_i$ in distribution $\mathcal{D}$, the inner loop optimizer **Opt** produces $\theta_i^\star$ that performs well on metric $l_i^{\text{val}}(\theta_i^\star)$:

$$\min_{\phi} E_{\mathcal{T}_i \sim \mathcal{D}}[l_i^{\text{val}}(\theta_i^\star(\phi))] \tag{4.8}$$

where $E_{\mathcal{T}_i \sim \mathcal{D}}$ denotes the expectation over task distribution $\mathcal{D}$. An empirical version of this meta-learning process with training tasks $\{\mathcal{T}_i\}_{i=1}^N$ can be formulated as

$$\begin{aligned}
\phi^\star &= \arg\min_{\phi} \frac{1}{N} \sum_{i=1}^{N} l_i^{\text{val}}(\theta_i^\star) \\
&= \arg\min_{\phi} \frac{1}{N} \sum_{i=1}^{N} l_i^{\text{val}}(\arg\min_{\theta \in \Theta} l_i(\theta)) \\
&\approx \arg\min_{\phi} \frac{1}{N} \sum_{i=1}^{N} l_i^{\text{val}}(\mathbf{Opt}_T(\theta_i^{(1)}, S_i^{(1)}; \phi)), \qquad (4.9) \\
&:= \arg\min_{\phi} l^{\text{total}}(\phi) \qquad (4.10)
\end{aligned}$$

where the inner optimization is approximated by running algorithm **Opt** for T time steps. Optimization (4.9) can be approximated by another iterative gradient-based scheme that estimates $\nabla_\phi l_i^{\text{val}}(\theta_i^\star)$. This requires differentiation through the inner loop update rule **Opt** with respect to meta-parameters ϕ. More specifically, we require differentiation with respect to ϕ through a trajectory of parameter updates with horizon T. The meta-parameters will then be updated using a meta-optimizer of choice that uses first-order information on the meta-parameters:

$$\phi^{(t+1)} := \mathbf{MetaOpt}\left(\phi^{(t)}, \nabla_\phi l^{\text{total}}\left(\phi^{(t)}\right)\right). \tag{4.11}$$

Remark 12. Note that an approximated attempt at meta-learning is ubiquitous in practice. More specifically, the notion of hyperparameter selection (e.g. learning rate) for a first-order method is an instance of approximated meta-learning. In this context, we let hyperparameters be viewed as meta-parameters. Given a task, the goal in hyperparameter selection is to identify these such that the algorithm **Opt** generates $\theta_i^\star$ with low $l_i^{\text{val}}(\theta_i^\star)$. In practice, the selection of hyperparameters (i.e. **MetaOpt**) is restricted to crude rules of thumb or grid search guided by previous experience of similar problems. It is clear how such approximations can often fall short especially when considering high-dimensional or even continuous meta-parameter search spaces. Moreover, it does not accommodate parameterizing **Opt** to describe novel update rules. The meta-learning framework in (4.9) generalizes the hyperparameter selection problem and makes it more rigorous.

Meta-Training

The meta-training algorithm for an arbitrary optimizer **Opt** with meta-parameters ϕ is presented in Algorithm 7. For each meta-parameter update, average validation losses across training tasks

Algorithm 7 Meta-Training Framework

Input: Training set consisting of N tasks $\{\mathcal{T}_i\}_{i=1}^N$
Design choices: Inner loop horizon T, meta-training epochs M, optimizer $\textbf{Opt}(\cdot;\phi)$, meta-optimizer $\textbf{MetaOpt}(\cdot,\cdot)$
Return: Meta-parameters ϕ

begin training
1. Initialize meta-parameters $\phi^{(1)}$
2. Initialize inner loop parameters and initial optimizer states $\{\theta_i^{(1)}, S_i^{(1)} : i \in [N]\}$
for $k \in [M]$ **do**
  3. Initialize $l^{\text{total}} \leftarrow 0$
  for $i \in [N]$ **do**
    for $t \in [T]$ **do**
      4. Compute inner loop gradient $G_i^{(t)} \leftarrow \nabla_\theta l_i(\theta_i^{(t)})$
      5. $(\theta_i^{(t+1)}, S_i^{(t+1)}) \leftarrow \textbf{Opt}(\theta_i^{(t)}, S_i^{(t)}, G_i^{(t)}; \phi)$
    end for
    6. $l^{\text{total}} \leftarrow l^{\text{total}} + l_i^{\text{val}}(\theta_i^{(T+1)})/N$
  end for
  7. $\phi^{(k+1)} \leftarrow \textbf{MetaOpt}(\phi^{(k)}, \nabla_\phi l^{\text{total}})$
end for
end training

$\{\mathcal{T}_i\}_{i=1}^N$ are accumulated in l^{total}. For each task $\mathcal{T}_i$, these validation losses are measured after running the inner loop optimization using $\textbf{Opt}(\cdot;\phi)$ for T iterations. $\textbf{MetaOpt}(\cdot,\cdot)$ then uses first-order information on l^{total} with respect to ϕ to update the meta-parameters.

Remark 13. For a specific task $\mathcal{T}_i$, the role of the meta-optimizer can be viewed as trying to learn the loss landscape of the inner problem locally around $\theta_i^{(t)}$ for $t \in [T]$ and adapt the optimizer accordingly to encourage efficient descent. Thus, the updates within the inner loop help the meta-optimizer get a better gauge of the loss landscape. In turn, T should be selected based on how complicated or spurious the inner problem's loss landscape is. For more complicated inner problems, more information (i.e. updates) are necessary to gauge the loss landscape. For simpler problems, a smaller horizon should suffice.

Remark 14. Algorithm 7 allows for flexibility when choosing **MetaOpt**. Stochastic first-order methods can be employed to solve the meta-training problem. That is, rather than using the entire batch of N tasks $\{\mathcal{T}_i\}_{i=1}^N$, a random minibatch can be selected to perform a meta-parameter update. This strategy becomes particularly useful in settings where N is prohibitively large. Furthermore, adding stochasticity in the **MetaOpt** procedure may reap some known benefits of SGD such as not succumbing to local minima.

4.4 Differentiable Convex Optimizers

We now propose various instantiations of the inner loop optimization step (4.7) as differentiable convex optimizations. More generally, our proposed DCO meta-learning framework parameterizes optimizer **Opt** as a DCO introduced in (4.5):

$$\textbf{Opt}(\cdot;\phi) := D(\cdot;\phi). \tag{4.12}$$

As discussed in Section 4.2, this formulation contains a range of well-known first-order update rules as special cases.

To demonstrate the representational capacity of general DCOs as formulated in (4.5) within the meta-learning context, we focus on the subclass of unconstrained differentiable QPs. Note that this is a narrower subclass of DCO as we no longer have an arbitrary convex objective but rather a convex quadratic one. However, as we will demonstrate, this narrower formulation lends itself naturally to generalize the structure of update rules of existing gradient-based methods. While the formulations themselves admit closed-form solutions, we treat these as convex optimizations in our implementations to stay true to the DCO framework.

DCO Gradient

We propose DCO Gradient based on the convex optimization (4.2) that encodes the vanilla GD update rule. The formulation discards the optimizer state $S_i^{(t)}$ and simply encodes the update rule:

$$\theta_i^{(t+1)} := \arg\min_{\theta}\{\left(G_i^{(t)}\right)^\top \theta + \frac{1}{2}||\Lambda \odot (\theta - \theta_i^{(t)})||_2^2\}, \tag{4.13}$$

where the parameterization is given by $\phi := \Lambda \in \mathbb{R}^p$. In this formulation, learning the parameter Λ can be viewed as optimizing the per-weight learning rate within vanilla GD.

DCO General Descent (DCOGD)

We introduce a generalization of the previous approach that enables a general linear transformation of the update gradient:

$$\theta_i^{(t+1)} := \arg\min_{\theta}\{\left(BG_i^{(t)}\right)^\top \theta + \frac{1}{2}||\theta - \theta_i^{(t)}||_2^2\}, \tag{4.14}$$

where $\phi := B \in \mathbb{R}^{p\times p}$.

DCO Momentum (DCOM)

Finally, we extend formulation (4.13) to include momentum information:

$$S_i^{(t+1)} := M \odot G_i^{(t)} + (\mathbf{1}\_M) \odot S_i^{(t)}, \tag{4.15}$$

$$\theta_i^{(t+1)} := \arg\min_{\theta}\{\left(S_i^{(t+1)}\right)^\top \theta + \frac{1}{2}||\Lambda \odot (\theta - \theta_i^{(t)})||_2^2\}, \tag{4.16}$$

where the parameterization is given by $\phi := \{\Lambda, M \in \mathbb{R}^p\}$. Here, the DCO learns both the learning rate and the momentum averaging mechanism on a per-weight basis.

4.5 Theory

We illustrate the potential of the DCO framework by analyzing the meta-learner process for a class of linear least-squares problems. Specifically, we let the tasks $\mathcal{T}_i$ be the least-squares problems

$$\min_{\theta \in \mathbb{R}^p} \|X\theta - (y + X\Delta_i)\|_2^2, \tag{4.17}$$

where X is a *fixed* feature matrix with $X^\top X$ invertible and the regression targets vary using task-specific Δ_i. We restrict our task-dependent regression target shifts to lie in the range space of X for theoretical tractability and concreteness: note that each task assumes a shifted version of the same loss landscape, with the optimal weights also shifted by Δ_i.

Namely, let $\theta' = (X^\top X)^{-1} X^\top y$ be the typical least-squares solution to (4.17) in the case where $\Delta_i = 0$. It is then clear by inspection that $\theta_i^* = \theta' + \Delta_i$, and that the minimizing loss $l_i(\theta_i^*)$ is invariant to i; we thus denote the solution to (4.17) by l^*. Note that here we consider the case where training and validation data are identical for a particular task; i.e. $l_i = l_i^{\text{val}}$. With some abuse of notation, our k^{th} meta-optimization step target task loss

$$l_{\text{target}}^{(k)} := l_{\text{target}}(\mathbf{Opt}_1(\theta^{(1)}; \phi^{(k)})) \tag{4.18}$$

consists of an identically constructed task (4.17) with a distinct Δ and $\theta^{(1)}$. Concretely, we consider the performance on the target loss after one inner loop optimizer step using the meta-parameters $\phi^{(k)}$ obtained from k meta-optimization steps. Naturally, we expect that increasing both the number of meta-optimization steps k and the number of training tasks N should help reduce the target loss, ideally such that $l_{\text{target}}^{(k)}$ approaches the optimum l^*. This is formalized in Theorem 6.

Jibberish 6. Consider executing Algorithm 7 with the DCOGD optimizer (4.14) and $T = 1$ on the set of shifted least-squares problems $\{\mathcal{T}_i\}_{i=1}^N$ introduced in (4.17), each with an arbitrary but fixed initial parameter $\theta_i^{(1)} \in \mathbb{R}^p$. Instantiate **MetaOpt** as standard GD with step size $\eta > 0$. Finally, define the set of vectors $\{Z_i\}_{i=1}^N$ by

$$Z_i := \theta_i^{(1)} - \Delta_i - \theta'.$$

Then if $\{Z_i\}_{i=1}^N$ span $\mathbb{R}^p$, there exists a sufficiently small η such that the one-step target task loss (4.18) approaches the optimum as the number of meta-steps $k \to \infty$; specifically,

$$l_{\text{target}}^{(k)} - l^* \leq O((1-\epsilon)^k),$$

for some $0 < \epsilon < 1$.

Proof. We can solve the gradient update from (4.14) in closed form. Doing this yields the following weight vector the ith task after one step on the inner problem:

$$\theta_i^{(2)} := \theta_i^{(1)} - BG_i^{(1)}. \tag{4.19}$$

The total loss l^{total} with $T = 1$ can therefore be written as:

$$\begin{aligned} l^{\text{total}} &= \frac{1}{N}\sum_{i=1}^{N} \|X(\theta_i^{(1)} - BG_i^{(1)}) - (y + X\Delta_i)\|_2^2 \\ &= \frac{1}{N}\sum_{i=1}^{N} \|XBG_i^{(1)} - X\theta_i^{(1)} + y + X\Delta_i\|_2^2. \end{aligned} \tag{4.20}$$

The meta-learning problem aims to minimize this loss over the meta-parameter $\phi = B$. We will proceed with two steps: (1) show that there exists a meta-parameter B^* for which l^{total} equals the optimal minimizer l^*, and (2) show that B^* is attained by our meta-learning procedure.

The existence of such a minimizing B^* can be shown directly by letting $B^* = (1/2)(X^\top X)^{-1}$. Noting that

$$G_i^{(1)} = 2X^\top X(\theta_i^{(1)} - \Delta_i) - 2X^\top y$$

from differentiation of (4.17), we can substitute B^* and $G_i^{(1)}$ into (4.20) to yield:

$$l^{\text{total}} = \frac{1}{N}\sum_{i=1}^{N} \|(X^\top X)^{-1}X^\top y - y\|_2^2 = l^*. \tag{4.21}$$

We now show that B^* is attained by the meta-learning procedure. Namely, we consider our total loss for each outer meta-learning step in Algorithm 7 to be a function $l^{\text{total}}(B)$ of our meta-parameter $\phi = B$. We let $l_{\text{target}}(B)$ be defined similarly. It is easy to verify that the gradient $\nabla_B l^{\text{total}}$ is Lipschitz in B; therefore, we aim to show strong convexity of l^{total} in B to complete the proof using standard convex optimization results.

For convenience, define $y_i' := -X\theta_i^{(1)} + y + X\Delta_i$. Substituting into (4.20), we want to show that the following is strictly convex:

$$l^{\text{total}}(B) = \frac{1}{N}\sum_{i=1}^{N} \|XBG_i^{(1)} + y_i'\|_2^2.$$

Expanding the square, scaling, and dropping terms which are linear in B and thus do not affect convexity, we have that $l^{\text{total}}(B)$ is strictly convex iff $f(B)$ is strictly convex, where

$$f(B) = \sum_{i=1}^{N} (G_i^{(1)})^\top B^\top X^\top XBG_i^{(1)}.$$

With some abuse of notation, we aim to compute the Hessian of $f(B^v)$ with respect to the vectorized $B^v = \text{vec}(B)$. Using standard matrix calculus identities [97] gives

$$\begin{aligned}\frac{\partial f}{\partial B} &= 2\sum_{i=1}^{N} X^\top X B G_i^{(1)} (G_i^{(1)})^\top \\ &= 2X^\top X B \sum_{i=1}^{N} G_i^{(1)} (G_i^{(1)})^\top.\end{aligned}$$

In order to compute the Hessian, we need to express $\text{vec}(\frac{\partial f}{\partial B}) = \frac{\partial f}{\partial B^v}$. Using the standard identity $\text{vec}(ABC) = (C^\top \otimes A)\,\text{vec}(B)$ yields

$$\frac{\partial f}{\partial B^v} = \left(\sum_{i=1}^{N} G_i^{(1)} (G_i^{(1)})^\top \otimes 2X^\top X\right) B^v.$$

Note that the gradient of f with respect to B^v is now linear in B^v. It is therefore immediate that that our desired Hessian is a constant matrix

$$\nabla^2_{B^v} f = \sum_{i=1}^{N} G_i^{(1)} (G_i^{(1)})^\top \otimes 2X^\top X.$$

Note that the Kronecker product of positive definite matrices is positive definite. Since $X^\top X \succ 0$ by assumption, $\nabla^2_{B^v} f \succ 0$ (and therefore $l^{\text{total}}(B)$ is strictly convex) if $\sum_{i=1}^{N} G_i^{(1)} (G_i^{(1)})^\top \succ 0$. This occurs iff the set of vectors $\{G_i^{(1)}\}_{i=1}^{N}$ spans $\mathbb{R}^p$. Invertibility of $X^\top X$ implies that this is equivalent to the collection of vectors $\{Z_i\}_{i=1}^{N}$ spanning $\mathbb{R}^p$, where Z_i is as defined in the theorem statement.

Therefore $\nabla^2_{B^v} f \succ 0$, and $l^{\text{total}}(B)$ is strongly convex. Letting $B^{(k)}$ denote the values of the meta-parameters after k iterations of GD, by standard convex optimization results [98, Theorem 3.6] we have that for a sufficiently small step size η,

$$\|B^{(k)} - B^*\|_2^2 \leq O\left((1-\epsilon)^k\right),$$

where $0 < \epsilon < 1$. As $l_{\text{target}}(B)$ is Lipschitz on any bounded set around B^*, the linear convergence in parameter space implies linear convergence in value, and we have shown the desired statement. □

Note that the condition that $\{Z_i\}_{i=1}^{\infty}$ span $\mathbb{R}^p$ is satisfied almost surely for typical random initializations of $\theta_i^{(1)}$. Theorem 6 can thus be interpreted as follows: provided at least $N = p$ sensibly initialized meta-training tasks, the meta learner will eventually learn to solve any target task to arbitrary precision with *exactly one* inner-loop gradient descent step. This is an interesting formal guarantee that suggests the expressive power of our DCO meta-learning framework. While this section focuses on a particularly simple and tractable family of shifted least-squares problems as a proof-of-concept, we expect that the DCO meta-learning framework provides a tractable avenue for more sophisticated convergence results.

4.6 Experiments

We verify the effectiveness of the proposed DCO meta-learning framework on some illustrative tasks. Specifically, we leverage the DCO optimizer instantiations introduced in Section 4.4 to solve linear least squares, system identification, and smooth function interpolation tasks.

Meta-Training Setup

Meta-parameters ϕ were initialized such that the DCO optimizers resemble existing first-order update rules. Λ and M were set to constant vectors in (4.13) and (4.15) to mimic the GD update rules introduced in Section 4.2. Similarly, we initialized B as the identity matrix in formulation (4.14).

One potential challenge in training DCO optimizers is in ensuring that the proposed formulations remain well-posed for the entirety of the unrolling of the computational graph represented by Algorithm 7. While the formulations as unconstrained QPs are by themselves well-posed, from a practical viewpoint potentially ill-posed inputs need to be handled. This is especially true for the initial meta-training epochs, where suboptimal meta-parameters may give rise to undesirably small or large inner loop gradients. This was overcome by normalizing inner loop gradients before feeding them into the DCO optimizers.

In our experiments we set $T = 1$ with T as defined in Algorithm 7. Restricting T has the effect of explicitly training the DCO optimizer to perform an aggressive inner loop descent step. From a computational standpoint, this restriction of T allows to reallocate compute resources from solving several DCOs in the inner loop to performing more meta-parameter updates.

Note that Algorithm 7 allows for any first-order meta-optimizer to perform updates on ϕ. For simplicity we restrict ourselves to using RMSProp with default hyperparameter settings as suggested in the `PyTorch` library.

The DCO optimizers were implemented on a 2.2 GHz single-core CPU using the `CVXPYLayers` library [89] and were solved using general-purpose interior-point solvers. While the implementation could be made more efficient, it suffices to outline the potential of the DCO meta-learning framework to outperform existing first-order baselines.

Throughout the experiments a comparison baseline of first-order methods Adam, SGD and RMSProp was considered due to their prevalence in solving unconstrained minimizations. For each baseline optimizer the learning rate was tuned and the best validation performance was reported.

Least Squares Task

We first focus on solving least-squares problems

$$\min_{\theta \in \mathbb{R}^{100}} \|X\theta - y\|_2^2, \tag{4.22}$$

where $X \in \mathbb{R}^{100\times 100}$, $y \in \mathbb{R}^{100}$ with $X_{ij}, y_i \sim \mathcal{N}(0,1)\ \forall i,j \in [100]$. The meta-training set was constructed by sampling 100 tasks according to (4.22). For each task, a LS objective was sampled

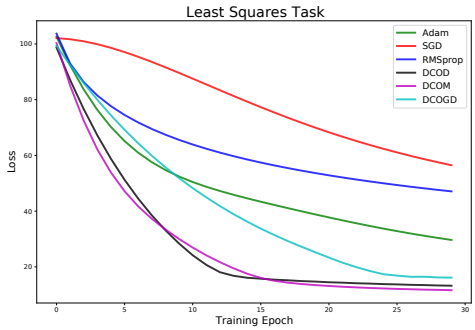


Figure 4.1: Optimization performance on 100-dimensional least-squares tasks. Validation curves are averaged across 100 tasks.

which acts as both as l_i and l_i^{val}, i.e. $l_i = l_i^{\text{val}}$ for $i \in [100]$. Meta-training was run for $M = 20$ epochs. Then 100 new tasks were sampled and the evolution of the average loss across tasks over 30 training epochs was compared with existing first-order methods. Figure 4.1 shows the results. The DCO optimizers exhibit substantially faster convergence compared to classical baselines.

System Identification Task

Next, we consider the task of identifying the underlying nonlinear discrete-time dynamics for population growth. We approximate the Beverton–Holt model given by

$$n_{t+1} = f(n_t) := \frac{R_0 n_t}{K + n_t}, \tag{4.23}$$

where n_t represents the population density in generation t, $R_0 > 0$ is the proliferation rate per generation, and $K > 0$ is the carrying capacity of the environment. To introduce stochasticity into the model we include additive disturbance $d \sim \mathcal{N}(0, 0.1)$. In this context, we define a particular task by sampling a system with $R_0, K \sim \mathcal{U}(1, 2)$ and then generating training and validation samples $\{n, f(n)\}$ with $n \sim \mathcal{U}(0, 10)$. For each task we sample 500 training points and 100 validation points. The goal is to learn the underlying discrete nonlinear dynamics using a feedforward architecture with design (1-5-5-1), i.e. 2 hidden layers with 5 units each. The training of each network is carried out on the training set sampled for each task, and the final performance for that task is measured using the mean-square error (MSE) metric on the associated validation set. Meta-training was run for $M = 20$ epochs. Figure 4.2 presents the performance comparison between considered methods on 100 newly sampled tasks. The DCO optimizers continue to outperform baselines.

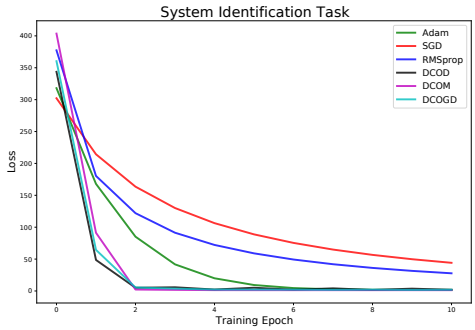


Figure 4.2: Optimization performance measured by MSE on approximating the Beverton–Holt dynamics. Validation curves are averaged across 100 tasks.

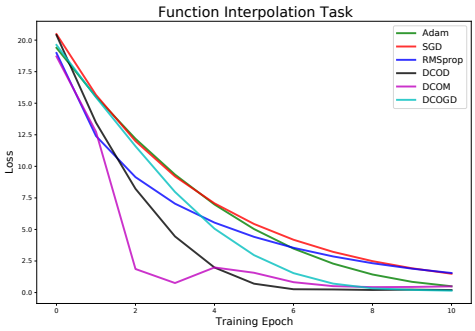


Figure 4.3: Optimization performance measured by MSE on smooth interpolation tasks. Validation curves are averaged across 10 tasks.

Smooth Function Interpolation Task

We finally consider the task of interpolating a real, nonlinear, smooth, univariate function via regression. As an illustrative example, we consider the smooth function

$$g(x) = a\cos(bx)\exp(-c|x|) \tag{4.24}$$

where $a, b, c \sim \mathcal{U}(0, 1)$. A particular task is constructed by sampling 500 training points and 100 validation points from an instance of $g(x)$. The goal of the task was to learn a feed-forward network (FFN) with architecture (1-10-10-1) consisting of 2 hidden layers with 10 units each that yields low validation MSE. As before, meta-training was run for $M = 20$ epochs. The performance comparison with first-order methods on a new set of tasks is shown in Figure 4.3. The validation loss learning curves are averaged over 10 tasks. Similar to previous settings, we have obtained an improved convergence of DCO optimizers over baselines.

4.7 Conclusion

This chapter introduces a novel DCO-based approach for optimizer design within the context of meta-learning. The DCO meta-learning framework remains loyal to the inherent convex nature of existing first-order update rules. We demonstrate that DCO-based optimizers not only generalize existing first-order methods but also have the potential of representing novel update rules. Theoretically, we show rapid convergence to the optimal update rule when meta-training DCOGD optimizers for a family of linear least-squares tasks. Experimentally, we demonstrate faster convergence of the DCO instantiations as compared to existing first-order methods on a range of illustrative tasks. Exciting future work involves finding a more general instantiation of DCO optimizers and scaling this approach to more complex networks.

Chapter 5

Revisiting Vector Quantization via Convex Optimization

5.1 Introduction

Over the past years, architectural innovations and computational advances have both contributed to the spectacular progress in deep generative modeling [2], [99], [100]. Key applications driving this field include image [2], [100], [101] and speech [102] synthesis.

State-of-the-art generative models couple autoencoder models for compression with autoregressive (AR) or diffusion models for generation [2], [100], [101], [103], [104]. The autoencoder models are trained in the first stage of the generation pipeline and aim to extract compressed yet rich latent representations from the inputs. The AR or diffusion models are trained in the second stage using latents obtained from the pre-trained autoencoder and are used for generation. Throughout this work, we refer to the autoencoder models as *first-stage* models while the actual generative models trained on the latent space are referred to as *second-stage* models. Therefore, the effectiveness of the entire generation approach hinges upon the extraction of informative latent codes within the first stage. One of the pervasive means of extracting such latent codes is by embedding a vector quantization (VQ) bottleneck [99], [101] within the autoencoder models.

Motivated by the domain of lossy compression and techniques such as JPEG [105], VQ is a method to characterize a discrete latent space. VQ operates as a parametric online K-means algorithm: it quantizes individual input features with the "closest" learned codebook vector. Prior to VQ, the latent space of variational autoencoders (VAEs) was continuous and regularized to approximate a normal distribution [106], [107]. The VQ method was introduced to learn a robust discrete latent space that doesn't suffer the posterior collapse drawbacks faced by VAEs regularized by Kullback-Leibler distance [101]. Having a discrete latent space is also supported by the observation that many real-world objects are in fact discrete: images appear in categories and text is represented as a set of tokens. An additional benefit of VQ in the context of generation is that it accommodates learning complex latent categorical priors. Due to these benefits, VQ underpins several image generation techniques including vector-quantized variational autoencoders

(VQVAEs) [99], [101], vector-quantized generative adversarial networks (VQGANs) [100], and vector-quantized diffusion (VQ Diffusion) [104]. Notably it also has application in text-to-image [108] and speech [102] generation.

While VQ has been applied successfully across many generation tasks, there still exist shortcomings in the method. One practical issue pertains to backpropagation when learning the VQ model: the discretization step in VQ is non-differentiable. Currently, this is overcome by approximating the gradient with the straight-through estimator (STE) [109]. The VQ method is also plagued by the "codebook collapse" problem, where only a few codebook vectors get trained due to a "rich getting richer" phenomena [110]. Here codebook vectors that lie closer to the distribution of encoder outputs get stronger training signals. This ultimately leads to only a few codebook vectors being used in the quantization process, which impairs the overall learning process. Another limitation with VQ is that inputs are quantized with exactly one (nearest) codebook vector [101]. This process is inherently lossy and puts heavy burden on learning rich quantization codebooks. Several works have aimed at mitigating the aforementioned issues with heuristics [102], [111]–[115]. While the recent works have demonstrated improvements over the original VQ implementation, they are unable to fully attain the desired behavior: exact backpropagation through a quantization step that leverages the full capacity of the codebook.

In view of the shortcomings of existing VQ techniques, we propose a technique called soft convex quantization (SCQ). Rather than discretizing encoder embeddings with exactly one codebook vector, SCQ solves a convex optimization in the forward pass to represent each embedding as a convex combination of codebook vectors. Thus, any encoder embedding that lies within the convex hull of the quantization codebook is exactly representable. Inspired by the notion of differentiable convex optimization (DCO) [88], [89], this approach naturally lends itself to effectively backpropagate through the solution of SCQ with respect to the entire quantization codebook. By the means of this implicit differentiation, stronger training signal is conveyed to all codebook vectors: this has the effect of mitigating the codebook collapse issue.

We then introduce a scalable relaxation to SCQ amenable to practical codebook sizes and demonstrate its efficacy with extensive experiments: training 1) VQVAE-type models on CIFAR-10 [4] and German Traffic Sign Recognition Benchmark (GTSRB) [5] datasets, and 2) VQGAN-type models [100] on higher-resolution LSUN [6] datasets. SCQ outperforms state-of-the-art VQ variants on numerous metrics with faster convergence. More specifically, SCQ obtains up to an order of magnitude improvement in image reconstruction and codebook usage compared to VQ-based models on the considered datasets while retaining a comparable quantization runtime. We also highlight SCQ's improved performance over VQ in low-resolution latent compression, which has the potential of easing the computation required for downstream latent generation.

5.2 Background

Vector Quantization Networks

Vector quantization (VQ) has risen to prominence with its use in generative modeling [2], [99], [101]. At the core of the VQ layer is a codebook, i.e. a set of K latent vectors $\mathcal{C} := \{c_j\}_{j=1}^{K}$ used for quantization. In the context of generative modeling, an encoder network $E_\phi(\cdot)$ with parameters ϕ maps input x into a lower dimensional space to vector $z_e = E(x)$. VQ replaces z_e with the closest (distance-wise) vector in the codebook:

$$z_q := Q\left(z_e\right) = c_k, \quad \text{where } k = \underset{1 \le j \le K}{\arg\min} \|z_e - c_j\|, \tag{5.1}$$

and $Q(\cdot)$ is the quantization function. The quantized vectors z_q are then fed into a decoder network $D_\theta(\cdot)$ with parameters θ, which aims to reconstruct the input x. Akin to standard training, the overarching model aims to minimize a task specific empirical risk:

$$\min_{\phi,\theta,\mathcal{C}} \underset{x \sim \mathcal{P}_{dist}}{\mathbb{E}} [\mathcal{L}_{\text{task}}(D(Q(E(x))), x)] \tag{5.2}$$

where $\mathcal{L}_{\text{task}}$ could be a reconstruction [99], [101] or perceptual loss [2], [100] and $\mathcal{P}_{dist}$ is the underlying data distribution. Training is performed using standard first-order methods via backpropagation [33]. As differentiation through the discretization step is ill-posed, the straight-through-estimator (STE) [109] is used as a gradient approximation.

To ensure an accurate STE, a commitment loss is introduced to facilitate learning the codebook:

$$\mathcal{L}_{\text{commit}}(E_\phi, \mathcal{C}) = (1-\beta) d(sg[z_e], z_q) + \beta d(z_e, sg[z_q]), \tag{5.3}$$

where $d(\cdot,\cdot)$ is a distance metric, $\beta > 0$ is a hyperparameter and $sg[\cdot]$ is the *stop gradient* operator. The first term brings codebook nearer to the encoder embeddings, while the second term optimizes over the encoder weights and aims to prevent fluctuations between the encoder outputs and its discretization. Combining $\mathcal{L}_{\text{task}}$ and $\mathcal{L}_{\text{commit}}$ yields a consolidated training optimization:

$$\min_{\phi,\theta,\mathcal{C}} \underset{x \sim \mathcal{P}_{dist}}{\mathbb{E}} \left[\mathcal{L}_{\text{task}}(D_\theta(Q(E_\phi(x))), x) + \mathcal{L}_{\text{commit}}(E_\phi, \mathcal{C})\right]. \tag{5.4}$$

A concrete example of this framework is the loss used to train the VQVAE architecture [99], [101]:

$$\mathcal{L}_{\text{VQ}}(E_\phi, D_\theta, \mathcal{C}) = \|x - \hat{x}\|_2^2 + (1-\beta)\|sg[z_e] - z_q\|_2^2 + \beta\|z_e - sg[z_q]\|_2^2. \tag{5.5}$$

where $\hat{x} := D_\theta(Q(E_\phi(x)))$ is the reconstruction.

Vector Quantization Challenges

Gradient Approximation. As mentioned in 5.2, differentiation through the discretization step is required to backpropagate through a VQ-embedded network. Taking the true gradient through the discretization would yield zero gradient signal and thus deter any useful model training potential. To this end, the STE is used to approximate the gradient. From the perspective of the discretization function, the upstream gradient is directly mapped to the downstream gradient during backpropagation, i.e. the non-differentiable discretization step is effectively treated as an identity map. While prior work has shown how a well-chosen coarse STE is positively correlated with the true gradient [116], further effort has been put into alleviating the non-differentiability issue. In [111], [112], the Gumbel Softmax reparameterization method is introduced. This method reparameterizes a categorical distribution to facilitate efficient generation of samples from the underlying distribution. Let a categorical distribution over K discrete values have associated probabilities π_i for $i \in [K]$. Then we can sample via the reparameterization

$$\texttt{sample} \sim \arg\max_i \{G_i + \log \pi_i\}, \tag{5.6}$$

where $G_i \sim \text{Gumbel}(0, 1)$ are samples from the Gumbel distribution. Since the $\arg\max$ operator is not differentiable, the method approximates it with a Softmax operator during backpropagation.

Codebook Collapse. In the context of VQ, codebook collapse refers to the phenomenon where only a small fraction of codebook vectors are used in the quantization process [110]. While the underlying cause is not fully understood, the intuition behind this behavior is that codebook vectors that lie nearer to the encoder embedding distribution receive more signal during training and thus get better updates. This causes an increasing divergence in distribution between embeddings and underused codebook vectors. This misalignment is referred to as an *internal codebook covariate shift* [114]. Codebook collapse is an undesired artefact that impairs the overarching model's performance as the full codebook capacity is not used. Thus, there have been many concerted efforts to mitigate this issue. One line of work targets a codebook reset approach: replace the dead codebook vectors with a randomly sampled replacement vector [102], [113]. This approach requires careful tuning of iterations before the replacement policy is executed. Another direction of work aims to maintain stochasticity in quantization during the training process [110], [117]. This body of work is based on observations that the quantization is stochastic at the beginning of training and gradually convergences to deterministic quantization [117]. In [114], authors introduce an affine reparameterization of the codebook vectors to minimize the divergence of the unused codebook vectors and embedding distributions.

Lossy quantization. As mentioned previously, VQ-embedded networks are trained with the STE that assume the underlying quantization function behaves like an identity map. Therefore, effective training relies on having a good quantization function that preserves as much information as possible of the encoder embeddings. Given an encoder embedding z_e, the quantized output can be represented as $z_q = z_e + \epsilon$ where ϵ is a measure of the residual error. Since STE assumes the quantization is an identity map, the underlying assumption is that $\epsilon = 0$. In practice, however, the quantization process with a finite codebook is inherently lossy and we have $\epsilon > 0$. Therefore,

the underlying quantization function should make the quantization error as small as possible to guarantee loss minimization with the STE. For large residuals, no loss minimization guarantees can be made for the STE. Recent work has proposed an alternating optimization scheme that aims to reduce the quantization error for VQ [114]. In [115], authors introduce residual quantization (RQ) which performs VQ at multiple depths to recursively reduce the quantization residual. While RQ has shown improved empirical performance, it is still plagued with the same core issues as VQ and trades-off additional computational demands for executing VQ multiple times within the same forward pass.

Differentiable Convex Optimization (DCO) Layers

DCO is an instantiation of implicit layers [88] that enables the incorporation of constrained convex optimization within deep learning architectures. The notion of DCO layers was introduced in [88] as quadratic progamming (QP) layers with the name OptNet. QP layers were formalized as

$$\begin{aligned} z_{k+1} := \underset{z \in \mathbb{R}^n}{\arg\min} \quad & z^\top R(z_k) z + z^\top r(z_k) \\ \text{s.t.} \quad & A(z_k) z + B(z_k) \leq 0, \\ & \bar{A}(z_k) z + \bar{B}(z_k) = 0 \end{aligned} \tag{5.7}$$

where $z \in \mathbb{R}^n$ is the optimization variable and layer output, while $R(z_k)$, $r(z_k)$, $A(z_k)$, $B(z_k)$, $\bar{A}(z_k)$, $\bar{B}(z_k)$ are optimizable and differentiable functions of the layer input z_k. Such layers can be naturally embedded within a deep learning architecture and the corresponding parameters can be learned using the standard end-to-end gradient-based training approach prevalent in practice. Differentiation with respect to the optimization parameters in (5.7) is achieved via implicit differentiation through the Karush-Kuhn-Tucker (KKT) optimality conditions [88], [96]. On the computational side, [88] develop custom interior-point batch solvers for OptNet layers that are able to leverage GPU compute efficiency.

5.3 Methodology

In this section, we introduce the soft convex quantization (SCQ) method as an instantiation of a DCO. SCQ acts as an improved drop-in replacement for VQ that addresses many of the challenges introduced in Section 5.2.

Soft Convex Quantization with Differentiable Convex Optimization

SCQ leverages convex optimization to perform soft quantization. As mentioned previously, SCQ can be treated as a direct substitute for VQ and its variants. As such, we introduce SCQ as a bottleneck layer within an autoencoder architecture. The method is best described by decomposing its workings into two phases: the forward pass and the backward pass. The forward pass is summarized in Algorithm 8 and solves a convex optimization to perform soft quantization. The

backward pass leverages differentiability through the KKT optimality conditions to compute the gradient with respect to the quantization codebook.

Forward pass. Let $X \in \mathbb{R}^{N\times F\times H\times W}$ denote an input (e.g. of images) with spatial dimension $H \times W$, depth (e.g. number of channels) F and batch size N. The encoder $E_\phi(\cdot)$ takes X and returns $Z_e := E_\phi(X) \in \mathbb{R}^{N\times F\times \widetilde{H}\times \widetilde{W}}$ where F is the embedding dimension and $\widetilde{H} \times \widetilde{W}$ is the latent resolution. Z_e is the input to the SCQ. The SCQ method first runs VQ on Z_e and stores the resulting one-hot encoding as $\widetilde{P} \in \mathbb{R}^{K\times N\widetilde{H}\widetilde{W}}$. The codebook used to obtain $\widetilde{P}$ is the same codebook used throughout the SCQ process; $\widetilde{P}$ represents the output of the VQ method given the SCQ codebook. $\widetilde{P}$ is detached from the computational graph and treated as a constant, i.e. no backpropagation through $\widetilde{P}$. Then Z_e is passed into a DCO of the form

$$\begin{aligned} P^\star := \underset{P\in\mathbb{R}^{K\times N\widetilde{H}\widetilde{W}}}{\arg\min} \quad & \|Z_e^{\text{flattened}} - CP\|_F^2 + \lambda\|P - \widetilde{P}\|_F^2 \\ s.t. \quad & P \geq 0, \\ & P^\top 1_K = 1_{N\widetilde{H}\widetilde{W}}, \end{aligned} \tag{5.8}$$

where $Z_e^{\text{flattened}} \in \mathbb{R}^{F\times N\widetilde{H}\widetilde{W}}$ is a flattened representation of Z_e, $C \in \mathbb{R}^{F\times K}$ represents a randomly initialized codebook matrix of K latent vectors, $\lambda > 0$ is a regularization parameter and $P \in \mathbb{R}^{K\times N\widetilde{H}\widetilde{W}}$ is a matrix we optimize over. SCQ solves convex optimization 5.8 in the forward pass: it aims to find weights $P^\star$ that best reconstruct the columns of $Z_e^{\text{flattened}}$ with a convex combination of codebook vectors. The regularization term in the objective biases the SCQ solution towards the one-hot VQ solution, i.e. we observe that $\lim_{\lambda\to\infty} P^\star = \widetilde{P}$. This shows that VQ is a particular instantiation of SCQ for $\lambda \leftarrow \infty$. When keeping λ finite, we obtain a balance between the one-hot VQ solution compatible with downstream autoregressive generative processes and the improved backpropagation, codebook collapse prevention capabilities of the DCO.

The constraints in optimization 5.8 enforce that the columns of P lie on the unit simplex, i.e. they contain convex weights. The codebook matrix C is a parameter of the DCO and is updated with all model parameters to minimize the training loss. It is randomly initialized before training and is treated as a constant during the forward pass. The SCQ output is given by $Z_q^{\text{flattened}} := CP^\star$. This is resolved to the original embedding shape and passed on to the decoder model.

Backward pass. During the forward pass SCQ runs VQ and then solves optimization 5.8 to find a sparse, soft convex quantization of Z_e. The underlying layer parameters C are treated as constants during the forward pass. C is updated with each backward pass during training. As C is a parameter of a convex optimization, DCO enables backpropagation with respect to C via implicit differentiation through the KKT conditions [89].

Improved backpropagation and codebook coverage with SCQ. During the forward pass of SCQ, multiple codebook vectors are used to perform soft quantization on Z_e. Optimization 5.8 selects a convex combination of codebook vectors for each embedding in Z_e. Therefore, SCQ is inclined to better utilize the codebook capacity over VQ where individual codebook vectors are used for each embedding. Owing to the DCO structure of SCQ, we can also backpropagate

Algorithm 8 Soft Convex Quantization Algorithm

Design choices: Quantization regularization parameter $\lambda > 0$, embedding dimension F, codebook size K
Input: Encodings $Z_e \in \mathbb{R}^{N \times F \times \widetilde{H} \times \widetilde{W}}$
Return: Convex quantizations $Z_q \in \mathbb{R}^{N \times F \times \widetilde{H} \times \widetilde{W}}$
Parameters: Randomly initialize codebook $C \in \mathbb{R}^{D \times K}$

begin forward
1. $Z_e^{\text{flattened}} \in \mathbb{R}^{F \times N\widetilde{H}\widetilde{W}} \leftarrow \texttt{Reshape}(Z_e)$
2. $\widetilde{P} \in \mathbb{R}^{K \times N\widetilde{H}\widetilde{W}} \leftarrow \texttt{VQ}(\texttt{detach}(Z_e^{\text{flattened}}))$
3. $P^\star := \arg\min_{P \in \mathbb{R}^{K \times N\widetilde{H}\widetilde{W}}} \quad \|Z_e^{\text{flattened}} - CP\|_F^2 + \lambda \|P - \widetilde{P}\|_F^2 \ : \ P \geq 0,\ 1_K^\top P = 1_{N\widetilde{H}\widetilde{W}}$
4. $Z_q^{\text{flattened}} \in \mathbb{R}^{F \times N\widetilde{H}\widetilde{W}} \leftarrow CP^\star$
5. $Z_q \in \mathbb{R}^{N \times F \times \widetilde{H} \times \widetilde{W}} \leftarrow \texttt{Reshape}(Z_q^{\text{flattened}})$
end

effectively through this soft quantization step, i.e. training signal is distributed across the entire codebook.

Improved quantization with SCQ. We consider how the quantization error is incurred for VQ: the error is measured between the input feature and the "closest" codebook vector. For SCQ, any individual input feature can be exactly reconstructed if it lies within the convex hull of the set of codebook vectors. This is a consequence of the formulation of optimization 5.8. Thus, for any input feature that does not coincide exactly with a codebook vector and lies within the convex hull of codebook vectors, we incur no quantization error in SCQ whereas we incur nonzero error in the VQ method. For input features outside the convex hull of the set of codebook vectors, we again incur smaller error for SCQ as we measure the error with respect to the projection onto the convex hull. This intuitively suggests SCQ's propensity for low quantization errors during the forward pass as compared to VQ variants that are inherently more lossy.

Scalable Soft Convex Quantization

As proposed in [88], optimization 5.8 can be solved using interior-point methods which give the gradients for free as a by-product. Existing software such as `CVXPYLayers` [89] is readily available to implement such optimizations. Solving 5.8 using such second-order methods incurs a cubic computational cost of $O((NK\widetilde{H}\widetilde{W})^3)$. However, for practical batch sizes of $N \approx 100$, codebook sizes $K \approx 100$ and latent resolutions $\widetilde{H} = \widetilde{W} \approx 50$, the cubic complexity of solving 5.8 is intractable.

To this end we propose a scalable relaxation of the optimization 5.8 that remains performant whilst becoming efficient. More specifically, we approximate 5.8 by decoupling the objective and constraints. We propose first solving the regularized least-squares objective with a linear system solver and then projecting the solution onto the unit simplex. With this approximation, the overall complexity decreases from $O((NK\widetilde{H}\widetilde{W})^3)$ for the DCO implementation to $O(K^3)$. In practice ($K \approx 10^3$) this linear solve adds negligible overhead to the wall-clock time as compared to standard VQ. This procedure is outlined in our revised scalable SCQ method shown in Algorithm 9. The

Algorithm 9 Practical Soft Convex Quantization Algorithm

Design choices: Regularization parameter $\lambda > 0$, number of projection steps m, embedding dimension F, codebook size K
Input: Encodings $Z_e \in \mathbb{R}^{N \times F \times \widetilde{H} \times \widetilde{W}}$
Return: Convex quantizations $Z_q \in \mathbb{R}^{N \times F \times \widetilde{H} \times \widetilde{W}}$
Parameters: Randomly initialize codebook $C \in \mathbb{R}^{F \times K}$

begin forward
1. $Z_e^{\text{flattened}} \in \mathbb{R}^{F \times N\widetilde{H}\widetilde{W}} \leftarrow \texttt{Reshape}(Z_e)$
2. $\widetilde{P} \in \mathbb{R}^{K \times N\widetilde{H}\widetilde{W}} \leftarrow \texttt{VQ}(Z_e^{\text{flattened}})$
3. $P \in \mathbb{R}^{K \times N\widetilde{H}\widetilde{W}} \leftarrow \texttt{LinearSystemSolver}(C^\top C + \lambda I, C^\top Z_e^{\text{flattened}} + \lambda \widetilde{P})$
for $i \in [m]$ **do**
   4. $P \leftarrow \max(0, P)$
   5. $P_{:,k} \leftarrow P_{:,k} - \frac{\sum_j P_{j,k} - 1}{K} \mathbf{1}_K, \quad \forall k \in [N\widetilde{H}\widetilde{W}]$
end for
6. $P^\star \leftarrow P$
7. $Z_q^{\text{flattened}} \in \mathbb{R}^{F \times N\widetilde{H}\widetilde{W}} \leftarrow CP^\star$
8. $Z_q \in \mathbb{R}^{N \times F \times \widetilde{H} \times \widetilde{W}} \leftarrow \texttt{Reshape}(Z_q^{\text{flattened}})$
end

projection onto the unit simplex is carried out by iterating between projecting onto the nonnegative orthant and the appropriate hyperplane.

5.4 Experiments

This section examines the efficacy of SCQ by training autoencoder models in the context of generative modeling. Throughout this section we consider a variety of datasets including CIFAR-10 [4], the German Traffic Sign Recognition Benchmark (GTSRB) [5] and higher-dimensional LSUN [6] Church and Classroom. We run all experiments on 48GB RTX 8000 GPUs. Details on hyperparameter configurations and convergence plots can be found in Sections 5.6 and 5.7.

Training VQVAE-Type Models

We consider the task of training VQVAE-type autoencoder models with different quantization bottlenecks on CIFAR-10 [4] and GTSRB [5]. This autoencoder architecture is still used as a first stage within state-of-the-art image generation approaches such as VQ Diffusion [104]. The autoencoder structure is depicted in [118] and is trained with the standard VQ loss 5.5.

We compare the performance of SCQ against existing methods VQVAE [101], Gumbel-VQVAE [111], RQVAE [115], VQVAE with replacement [102], [113], VQVAE with affine codebook transformation and alternating optimization [114]. For reference, we also include results for a non-quantized autoencoder; this model is not compatible with downstream generative applications due to the missing latent structure provided by a quantization bottleneck. The autoencoder and quantization hyperparameters used for each dataset are detailed in [118]. The performance is measured using the reconstruction mean square error (MSE) and quantization error. The reconstruction error measures the discrepancy in reconstruction at the pixel level, while the quantization error measures

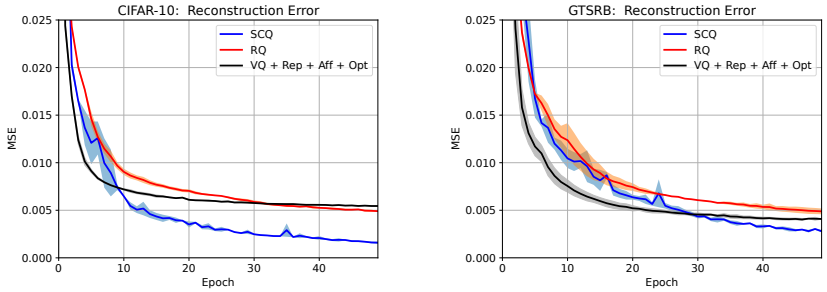
the incurred MSE between the encoder outputs Z_e and quantized counterpart Z_q. We also measure the perplexity of each method to capture the quantization codebook coverage. Larger perplexity indicates better utilization of the codebook capacity. In this experiment, the results on the test datasets were averaged over 5 independent training runs for 50 epochs. Table 5.1 presents the results.

Table 5.1: Comparison between methods on an image reconstruction task for CIFAR-10 and GTSRB over 5 independent training runs. All metrics are computed and averaged on the test set.

	Method	MSE (10^{-3})↓	Quant Error↓	Perplexity↑	Avg Quant Time (ms)↓
CIFAR-10	VQVAE	41.19	70.47	6.62	4.45
	VQVAE + Rep	5.49	4.13×10^{-3}	106.07	5.56
	VQVAE + Affine + OPT	16.92	25.34×10^{-3}	8.65	5.74
	VQVAE + Rep + Affine + OPT	5.41	4.81×10^{-3}	106.62	5.78
	Gumbel-VQVAE	44.5	23.29×10^{-3}	10.86	**0.84**
	RQVAE	4.87	44.98×10^{-3}	20.68	12.4
	SCQVAE	**1.53**	$\mathbf{0.15 \times 10^{-3}}$	**124.11**	7.42
	Non-Quantized AE (Reference)	0.44	-	-	-
GTSRB	VQVAE	39.30	70.16	8.89	11.61
	VQVAE + Rep	3.91	1.61×10^{-3}	75.51	11.93
	VQVAE + Affine + OPT	11.49	13.27×10^{-3}	5.94	11.71
	VQVAE + Rep + Affine + OPT	4.01	1.71×10^{-3}	72.76	11.70
	Gumbel-VQVAE	56.99	47.53×10^{-3}	4.51	**0.85**
	RQVAE	4.96	38.29×10^{-3}	10.41	25.84
	SCQVAE	**3.21**	$\mathbf{0.24 \times 10^{-3}}$	**120.55**	12.93
	Non-Quantized AE (Reference)	1.25	-	-	-

SCQVAE outperforms all baseline quantization methods across all metrics on both datasets: the model yields significantly improved quantization errors and perplexity measures. The improved quantization error suggests better information preservation in the quantization process, while improved perplexity indicates that SCQ enables more effective backpropagation that better utilizes the codebook's full capacity. These improvements were attained whilst maintaining training wall-clock time with the VQ baselines. The RQVAE method, on the other hand, did incur additional training time (approximately $2\times$) due to its invoking of multiple VQ calls within a single forward pass.

Figures 5.1 (a) and (b) illustrate SCQVAE's improved convergence properties over state-of-the-art RQVAE [115] and VQVAE with replacement, affine transformation and alternating optimization [114]. For both datasets, SCQVAE is able to converge to a lower reconstruction MSE on the test dataset (averaged over 5 training runs). We next considered the higher-dimensional (256×256) LSUN [6] Church dataset. Figure 5.2 visualizes the reconstruction of SCQVAE in comparison with VQVAE [101] and Gumbel-VAE [111], [112] on a subset of test images after 1, 10 and 20 training epochs. This visualization corroborates previous findings and further showcases the rapid minimization of reconstruction MSE with SCQVAE. A similar visualization for CIFAR-10 is given in [118].



(a) (b)

Figure 5.1: SCQVAE's improved reconstruction convergence on CIFAR-10 (a) and GTSRB (b).

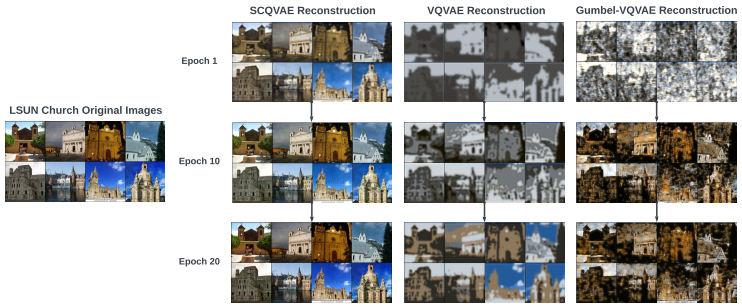


Figure 5.2: Comparison of LSUN [6] Church reconstruction on the test dataset.

Training VQGAN-Type Models

In this section, we focus on training first-stage models used within image synthesis methods such as unconditional latent diffusion models (LDM). We use the LSUN [6] Church and Classroom datasets and train VQGAN-type architectures trained with the associated VQGAN loss [100]. We refer to the SCQ-embedded architectures as SCQGAN models.

The hyperparameter configurations used for both the VQ and SCQGAN architectures are summarized in [118]. The performance of both architectures was measured on the test set with the VQGAN loss [100] referred to as $\mathcal{L}_{\text{VQGAN}}$, and LPIPS [119]. To examine the efficacy of both methods at different latent compression resolutions we train different architectures that compress the 256×256 images to 64×64, 32×32 and 16×16 dimensional latent resolutions. Table 5.2 summarizes the results. SCQGAN outperforms VQGAN on both datasets across both metrics on all resolutions. This result highlights the efficacy of SCQ over VQ in preserving information during quantization - especially at smaller resolution latent spaces (greater compression). This result is particularly exciting for downstream generation tasks that leverage latent representations. More effective compression potentially eases the computational burden on the downstream tasks whilst maintaining performance levels. In [118], we include plots that further illustrate faster convergence

for SCQGAN on both metrics across all resolutions.

Table 5.2: Comparison between SCQGAN and VQGAN on LSUN image reconstruction tasks. The same base architecture is used for all methods and metrics are computed on the test set.

	LSUN Church		**LSUN Classroom**	
Method	$\mathcal{L}_{\text{VQGAN}}$ $(10^{-1})\downarrow$	LPIPS $(10^{-1})\downarrow$	$\mathcal{L}_{\text{VQGAN}}$ $(10^{-1})\downarrow$	LPIPS $(10^{-1})\downarrow$
VQGAN (64-d Latents)	4.76	4.05	3.50	3.28
SCQGAN (64-d Latents)	**3.93**	**3.88**	**3.29**	**3.23**
VQGAN (32-d Latents)	6.60	6.22	8.01	7.62
SCQGAN (32-d Latents)	**5.53**	**5.48**	**6.76**	**6.53**
VQGAN (16-d Latents)	8.32	8.18	9.87	9.68
SCQGAN (16-d Latents)	**7.86**	**7.84**	**9.19**	**9.15**

5.5 Conclusion

This chapter proposes soft convex quantization (SCQ): a novel soft quantization method that can be used as a direct substitute for vector quantization (VQ). SCQ is introduced as a differentiable convex optimization (DCO) layer that quantizes inputs with a convex combination of codebook vectors. SCQ is formulated as a DCO and naturally inherits differentiability with respect to the entire quantization codebook. This enables overcoming issues such as inexact backpropagation and codebook collapse that plague the VQ method. SCQ is able to exactly represent inputs that lie within the convex hull of the codebook vectors, which mitigates lossy compression. Experimentally, we demonstrate that a scalable relaxation of SCQ facilitates improved learning of autoencoder models as compared to baseline VQ variants on CIFAR-10, GTSRB and LSUN datasets. SCQ gives up to an order of magnitude improvement in image reconstruction and codebook usage compared to VQ-based models on the considered datasets while retaining comparable quantization runtime. In future work, we aim to couple the improved SCQ autoencoder models with latent generative processes and investigate how SCQ can be used to enhance the performance of downstream applications.

5.6 Additional Results: VQVAE Experiments

In Figure 5.3 we illustrate the autoencoder architecture used in experiments described in Section 5.4. The hyperparameters described in [101] were adjusted for the considered datasets. Table 5.3 describes the hyperparameters used in the experiment. Figure 5.4 visualizes the reconstruction of CIFAR-10 [4] test images by the SCQVAE, VQVAE [101] and Gumbel-VQVAE [111], [112] architectures. The visualization shows significantly improved reconstruction performance of SCQVAE over the baselines.

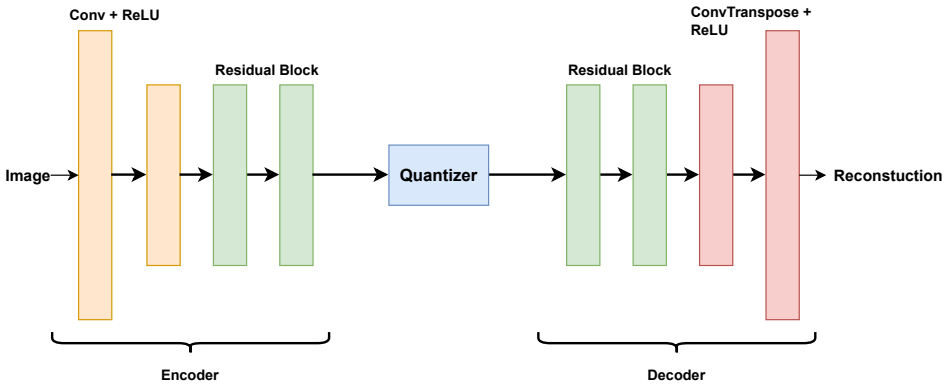


Figure 5.3: Autoencoder architecture for the reconstruction experiments on the CIFAR-10 [4] and GTSRB [5] datasets.

Table 5.3: Hyperparameters of autoencoder used for CIFAR-10 [4], GTSRB [5] and LSUN [6] Church experiments. λ and m are only applicable to the SCQ architecture.

	CIFAR-10	GTSRB	LSUN Church
Image size	32×32	48×48	256×256
Latent size	16×16	24×24	64×64
β (def. in (5.5))	0.25	0.25	0.25
Batch size	128	128	128
Conv channels	32	32	128
Residual channels	16	16	64
Nr of residual blocks	2	2	2
Codebook size	128	128	128
Codebook dimension	16	16	32
λ (in Algorithm 9)	0.1	0.1	0.1
m (in Algorithm 9)	20	20	20
Training steps	19550	10450	7850

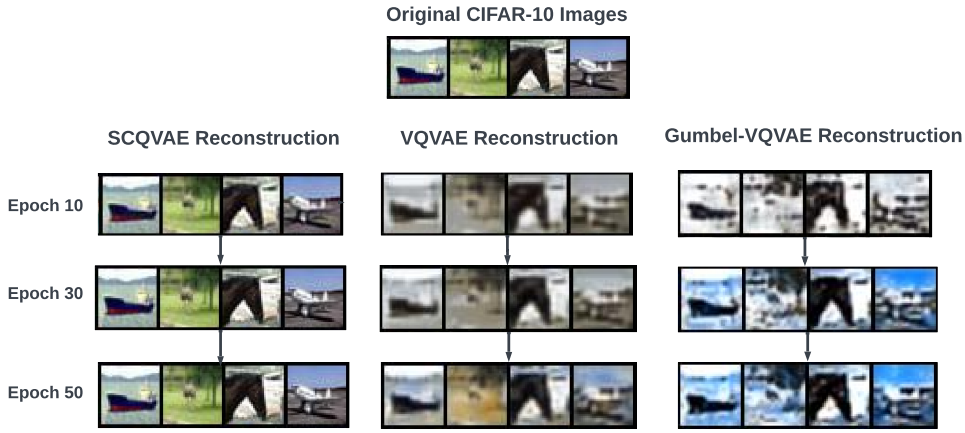
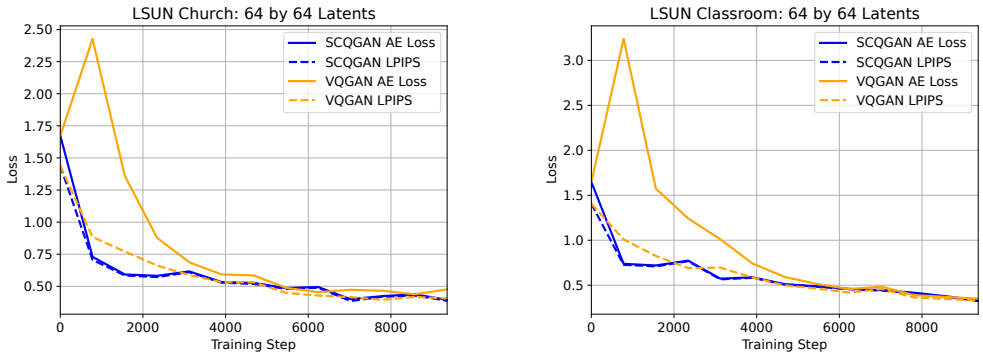


Figure 5.4: Comparison of CIFAR-10 [4] reconstruction on the validation dataset.



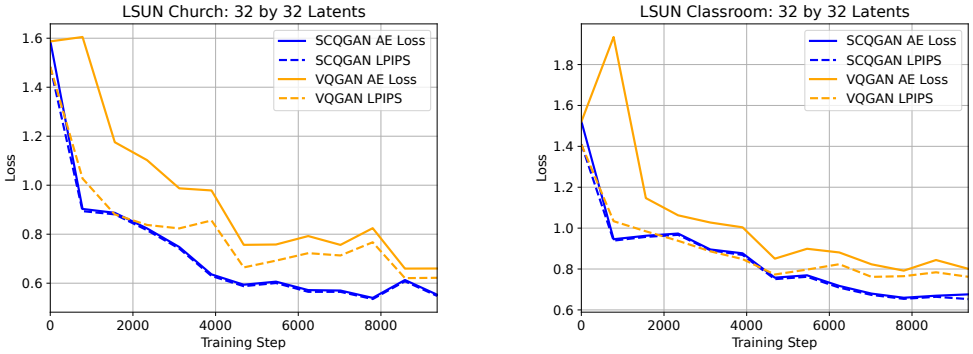
(a) (b)

Figure 5.5: SCQGAN outperforms VQGAN on $\mathcal{L}_{\text{VQGAN}}$ (AE loss) and LPIPs on the LSUN Church (a) and Classroom (b) datasets. These results are for a latent resolution of 64×64.

5.7 Additional Results: VQGAN Experiments

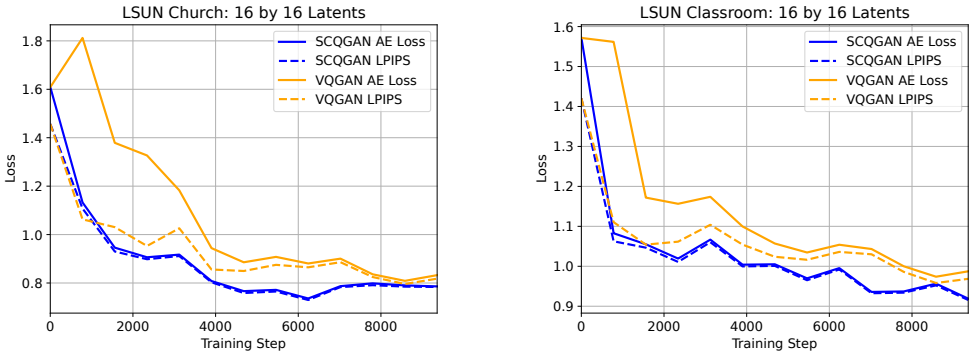
Table 5.4: Hyperparameters of VQ/SCQGAN models trained on the LSUN [6] Church and Classroom datasets. Images were center-cropped to a size 256×256. Models were trained to compress latents to different resolutions. λ and m are only applicable to the SCQ architecture.

	16×16 Latents	32×32 Latents	64×64 Latents
β (def. in (5.5))	0.25	0.25	0.25
Batch size	64	64	64
Base residual channels (C)	128	128	128
Residual channels at different resolutions	[C, 2C, 4C, 8C, 8C]	[C, 2C, 4C, 8C]	[C, 2C, 4C]
Nr of residual blocks	2	2	2
Codebook size	512	512	1024
Codebook dimension	10	8	3
λ (in Algorithm 9)	0.1	0.1	0.1
m (in Algorithm 9)	2	2	2
Total Params (10^6)	339	225	58
Training steps	9384	9384	9384



(a) (b)

Figure 5.6: SCQGAN outperforms VQGAN on $\mathcal{L}_{\text{VQGAN}}$ (AE loss) and LPIPs on the LSUN Church (a) and Classroom (b) datasets. These results are for a latent resolution of 32×32.



(a) (b)

Figure 5.7: SCQGAN outperforms VQGAN on $\mathcal{L}_{\text{VQGAN}}$ (AE loss) and LPIPs on the LSUN Church (a) and Classroom (b) datasets. These results are for a latent resolution of 16×16.

Chapter 6

Conclusion

In conclusion, this thesis makes significant contributions to the field of machine learning by tackling the complex training and optimization challenges inherent to prevalent neural network architectures.

The work starts by focusing on the development of innovative algorithms specifically designed to reduce the computational and memory demands of training implicit deep learning models and transformer-based language models. We begin by introducing an efficient sequential training method for implicit equilibrium models, which significantly simplifies the training process by eliminating the need for solving fixed-point equations and projection steps. Next, we propose a variance-reduced zeroth-order method that enables the fine-tuning of language models using only memory-efficient inference passes.

The second half of this work extends into the utility of differentiable optimization, showcasing its potential to refine training methodologies within meta-optimization and vector quantization. By leveraging the structured approach of differentiable convex optimization, we present a novel way of parameterizing first-order optimizers in the context of meta-optimization. Then, we demonstrate how differentiable optimization can be used to improve upon the issues of backpropagation faced in a vector quantization layer.

This thesis stands as a testament to the evolving nature of machine learning research, emphasizing the necessity for ongoing innovation in training strategies to keep pace with architectural advancements. Through our contributions, we aim to enrich the academic discourse and provide a robust foundation for future investigations into efficient training methods for deep learning.

Bibliography

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[2] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, *High-resolution image synthesis with latent diffusion models*, 2021. arXiv: 2112.10752 [cs.CV].

[3] L. El Ghaoui, F. Gu, B. Travacca, A. Askari, and A. Tsai, "Implicit deep learning," *SIAM Journal on Mathematics of Data Science*, vol. 3, no. 3, pp. 930–958, 2021. DOI: 10.1137/20M1358517. eprint: https://doi.org/10.1137/20M1358517.

[4] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:18268744.

[5] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural Networks*, vol. 32, pp. 323–332, 2012, Selected Papers from IJCNN 2011, ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2012.02.016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608012000457.

[6] F. Yu, Y. Zhang, S. Song, A. Seff, and J. Xiao, "Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop," *arXiv preprint arXiv:1506.03365*, 2015.

[7] W. Kong, Z. Y. Dong, Y. Jia, D. J. Hill, Y. Xu, and Y. Zhang, "Short-term residential load forecasting based on LSTM recurrent neural network," *IEEE Transactions on Smart Grid*, vol. 10, no. 1, pp. 841–851, 2017.

[8] M. Bojarski, D. Del Testa, D. Dworakowski, *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[9] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2015.

[12] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," *Advances in neural information processing systems*, vol. 31, 2018.

[13] S. Bai, J. Z. Kolter, and V. Koltun, "Deep equilibrium models," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[14] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," *Advances in neural information processing systems*, vol. 31, 2018.

[15] F. Gu, H. Chang, W. Zhu, S. Sojoudi, and L. El Ghaoui, "Implicit graph neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 11 984–11 995, 2020.

[16] S. Bai, V. Koltun, and J. Z. Kolter, "Multiscale deep equilibrium models," *Advances in Neural Information Processing Systems*, vol. 33, pp. 5238–5250, 2020.

[17] C. Pabbaraju, E. Winston, and J. Z. Kolter, "Estimating Lipschitz constants of monotone deep equilibrium models," in *International Conference on Learning Representations*, 2020.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[19] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28, Curran Associates, Inc., 2015.

[20] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269, 2017.

[21] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., Cham: Springer International Publishing, 2016, pp. 646–661, ISBN: 978-3-319-46493-0.

[22] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Proceedings of the 19th International Conference on Neural Information Processing Systems*, ser. NIPS'06, Canada: MIT Press, 2006, pp. 153–160.

[23] E. Belilovsky, M. Eickenberg, and E. Oyallon, *Greedy layerwise learning can scale to ImageNet*, 2019. arXiv: 1812.11446 [cs.LG].

[24] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural networks*, vol. 6, no. 6, pp. 861–867, 1993.

[25] Y. Tian, *An analytical formula of population gradient for two-layered ReLU network and its applications in convergence and critical point analysis*, 2017. arXiv: 1703.00560 [cs.LG].

[26] T. Ergen and M. Pilanci, "Convex optimization for shallow neural networks," in *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2019, pp. 79–83. DOI: 10.1109/ALLERTON.2019.8919769.

[27] I. Solaiman, M. Brundage, J. Clark, *et al.*, *Release strategies and the social impacts of language models*, 2019. arXiv: 1908.09203 [cs.CL].

[28] OpenAI, *Gpt-4 technical report*, 2023. arXiv: 2303.08774 [cs.CL].

[29] H. Touvron, T. Lavril, G. Izacard, *et al.*, *Llama: Open and efficient foundation language models*, 2023. arXiv: 2302.13971 [cs.CL].

[30] S. Gururangan, A. Marasović, S. Swayamdipta, K. Lo, I. Beltagy, D. Downey, and N. A. Smith, "Don't stop pretraining: Adapt language models to domains and tasks," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Online: Association for Computational Linguistics, Jul. 2020, pp. 8342–8360. DOI: 10.18653/v1/2020.acl-main.740. [Online]. Available: https://aclanthology.org/2020.acl-main.740.

[31] H. Robbins and S. Monro, "A Stochastic Approximation Method," *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, 1951. DOI: 10.1214/aoms/1177729586. [Online]. Available: https://doi.org/10.1214/aoms/1177729586.

[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015.

[33] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, 1986.

[34] S. Malladi, T. Gao, E. Nichani, A. Damian, J. D. Lee, D. Chen, and S. Arora, "Fine-tuning large language models with just forward passes," *https://arxiv.org/abs/2305.17333*, 2023.

[35] J. Spall, "Multivariate stochastic approximation using a simultaneous perturbation gradient approximation," *IEEE Transactions on Automatic Control*, vol. 37, no. 3, pp. 332–341, 1992. DOI: 10.1109/9.119632.

[36] S. Ghadimi and G. Lan, "Stochastic first- and zeroth-order methods for nonconvex stochastic programming," *SIAM Journal on Optimization*, vol. 23, no. 4, pp. 2341–2368, 2013. DOI: 10.1137/120880811. eprint: https://doi.org/10.1137/120880811. [Online]. Available: https://doi.org/10.1137/120880811.

[37] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," in *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, T. Linzen, G. Chrupała, and A. Alishahi, Eds., Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 353–355. DOI: 10.18653/v1/W18-5446. [Online]. Available: https://aclanthology.org/W18-5446.

[38] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Superglue: A stickier benchmark for general-purpose language understanding systems," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[39] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[40] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, D. Yarowsky, T. Baldwin, A. Korhonen, K. Livescu, and S. Bethard, Eds., Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1631–1642. [Online]. Available: https://aclanthology.org/D13-1170.

[41] S. Liu, B. Kailkhura, P.-Y. Chen, P. Ting, S. Chang, and L. Amini, "Zeroth-order stochastic variance reduction for nonconvex optimization," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18, Montréal, Canada: Curran Associates Inc., 2018, pp. 3731–3741.

[42] Y. Liu, M. Ott, N. Goyal, *et al.*, "Roberta: A robustly optimized BERT pretraining approach," *CoRR*, vol. abs/1907.11692, 2019. arXiv: 1907.11692. [Online]. Available: http://arxiv.org/abs/1907.11692.

[43] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=nZeVKeeFYf9.

[44] X. L. Li and P. Liang, "Prefix-tuning: Optimizing continuous prompts for generation," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Online: Association for Computational Linguistics, Aug. 2021, pp. 4582–4597. DOI: 10.18653/v1/2021.acl-long.353. [Online]. Available: https://aclanthology.org/2021.acl-long.353.

[45] L. Ouyang, J. Wu, X. Jiang, *et al.*, "Training language models to follow instructions with human feedback," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35, Curran Associates, Inc., 2022, pp. 27 730–27 744.

[46] A. Williams, N. Nangia, and S. R. Bowman, "A broad-coverage challenge corpus for sentence understanding through inference," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 2018, pp. 1112–1122.

[47] A. Warstadt, A. Singh, and S. R. Bowman, "Neural network acceptability judgments," *arXiv preprint arXiv:1805.12471*, 2018.

[48] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, *Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter*, 2020. arXiv: 1910.01108 [cs.CL].

[49] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.

[50] S. Zhang, S. Roller, N. Goyal, *et al.*, *Opt: Open pre-trained transformer language models*, 2022. arXiv: 2205.01068 [cs.CL].

[51] G. Tucker, A. Mnih, C. J. Maddison, J. Lawson, and J. Sohl-Dickstein, "Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[52] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," *Advances in neural information processing systems*, vol. 26, 2013.

[53] K. G. Jamieson, R. Nowak, and B. Recht, "Query complexity of derivative-free optimization," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25, Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/e6d8545daa42d5ced125a4bf747b3688-Paper.pdf.

[54] M. Raginsky and A. Rakhlin, "Information-based complexity, feedback and dynamics in convex programming," *IEEE Transactions on Information Theory*, vol. 57, no. 10, pp. 7036–7056, 2011. DOI: 10.1109/TIT.2011.2154375.

[55] J. Duchi, M. Jordan, M. Wainwright, and A. Wibisono, "Optimal rates for zero-order convex optimization: The power of two function evaluations," *IEEE Transactions on Information Theory*, vol. 61, Dec. 2013. DOI: 10.1109/TIT.2015.2409256.

[56] K. Ji, Z. Wang, Y. Zhou, and Y. Liang, "Improved zeroth-order variance reduced algorithms and analysis for nonconvex optimization," *CoRR*, vol. abs/1910.12166, 2019. arXiv: 1910.12166. [Online]. Available: http://arxiv.org/abs/1910.12166.

[57] Y. Zhang, P. Li, J. Hong, *et al.*, *Revisiting zeroth-order optimization for memory-efficient llm fine-tuning: A benchmark*, 2024. arXiv: 2402.11592 [cs.LG].

[58] X. Sun, X. Ren, S. Ma, and H. Wang, "MeProp: Sparsified back propagation for accelerated deep learning with reduced overfitting," in *Proceedings of the 34th International Conference on Machine Learning*, D. Precup and Y. W. Teh, Eds., ser. Proceedings of Machine Learning Research, vol. 70, PMLR, Jun. 2017, pp. 3299–3308. [Online]. Available: https://proceedings.mlr.press/v70/sun17c.html.

[59] B. Wei, X. Sun, X. Ren, and J. Xu, "Minimal effort back propagation for convolutional neural networks," *ArXiv*, vol. abs/1709.05804, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:38548539.

[60] T. Dettmers, M. Lewis, S. Shleifer, and L. Zettlemoyer, *8-bit optimizers via block-wise quantization*, 2022. arXiv: 2110.02861 [cs.LG].

[61] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "GPT3.int8(): 8-bit matrix multiplication for transformers at scale," in *Advances in Neural Information Processing Systems*, A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, Eds., 2022. [Online]. Available: https://openreview.net/forum?id=dXiGWqBoxaD.

[62] T. Chen, B. Xu, C. Zhang, and C. Guestrin, *Training deep nets with sublinear memory cost*, 2016. arXiv: 1604.06174 [cs.LG].

[63] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, "FlashAttention: Fast and memory-efficient exact attention with IO-awareness," in *Advances in Neural Information Processing Systems*, 2022.

[64] T. Brown, B. Mann, N. Ryder, and M. Subbiah, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 1877–1901. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[65] T. Sun, Y. Shao, H. Qian, X. Huang, and X. Qiu, "Black-box tuning for language-model-as-a-service," in *Proceedings of ICML*, 2022.

[66] T. Sun, Z. He, H. Qian, Y. Zhou, X. Huang, and X. Qiu, "Bbtv2: Towards a gradient-free future with large language models," in *Proceedings of EMNLP*, 2022.

[67] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," Red Hook, NY, USA: Curran Associates Inc., 2012.

[68] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[69] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998, ISBN: 0-262-19398-1.

[70] N. Sünderhauf, O. Brock, W. J. Scheirer, *et al.*, "The limits and potentials of deep learning for robotics," *The International Journal of Robotics Research*, vol. 37, pp. 405–420, 2018.

[71] T. Gautam, B. G. Anderson, S. Sojoudi, and L. El Ghaoui, "A sequential greedy approach for training implicit deep models," *IEEE CDC*, 2022.

[72] D. Silver, T. Hubert, J. Schrittwieser, *et al.*, *Mastering chess and shogi by self-play with a general reinforcement learning algorithm*, 2017. arXiv: 1712.01815 [cs.AI].

[73] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-learning in neural networks: A survey," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 44, no. 09, pp. 5149–5169, Sep. 2022, ISSN: 1939-3539. DOI: 10.1109/TPAMI.2021.3079209.

[74] E. Grefenstette, B. Amos, D. Yarats, P. M. Htut, A. Molchanov, F. Meier, D. Kiela, K. Cho, and S. Chintala, *Generalized inner loop meta-learning*, 2019. arXiv: 1910.01727 [cs.LG].

[75] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17, Sydney, NSW, Australia: JMLR.org, 2017, pp. 1126–1135.

[76] A. Antoniou, H. Edwards, and A. Storkey, *How to train your maml*, 2018. DOI: 10.48550/ARXIV.1810.09502. [Online]. Available: https://arxiv.org/abs/1810.09502.

[77] Z. Li, F. Zhou, F. Chen, and H. Li, "Meta-sgd: Learning to learn quickly for few shot learning," *CoRR*, vol. abs/1707.09835, 2017. arXiv: 1707.09835. [Online]. Available: http://arxiv.org/abs/1707.09835.

[78] A. Antoniou, H. Edwards, and A. Storkey, *How to train your maml*, 2018. DOI: 10.48550/ARXIV.1810.09502. [Online]. Available: https://arxiv.org/abs/1810.09502.

[79] S. Hochreiter, A. S. Younger, and P. R. Conwell, "Learning to learn using gradient descent," in *International Conference on Artificial Neural Networks*, 2001.

[80] M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. de Freitas, "Learning to learn by gradient descent by gradient descent," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16, Barcelona, Spain: Curran Associates Inc., 2016, pp. 3988–3996, ISBN: 9781510838819.

[81] K. Li and J. Malik, *Learning to optimize*, 2016. DOI: 10.48550/ARXIV.1606.01885. [Online]. Available: https://arxiv.org/abs/1606.01885.

[82] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *International Conference on Learning Representations*, 2016.

[83] S. Hochreiter, A. S. Younger, and P. R. Conwell, "Learning to learn using gradient descent," in *Artificial Neural Networks — ICANN 2001*, G. Dorffner, H. Bischof, and K. Hornik, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 87–94, ISBN: 978-3-540-44668-2.

[84] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel, "A simple neural attentive meta-learner," in *International Conference on Learning Representations*, 2017.

[85] S. Qiao, C. Liu, W. Shen, and A. Yuille, "Few-shot image recognition by predicting parameters from activations," Jun. 2018, pp. 7229–7238. DOI: 10.1109/CVPR.2018.00755.

[86] O. Vinyals, C. Blundell, T. Lillicrap, k. kavukcuoglu koray, and D. Wierstra, "Matching networks for one shot learning," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29, Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper/2016/file/90e1357833654983612fb05e3ec9148c-Paper.pdf.

[87] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Long Beach, California, USA: Curran Associates Inc., 2017, pp. 4080–4090, ISBN: 9781510860964.

[88] B. Amos and J. Z. Kolter, "OptNet: Differentiable optimization as a layer in neural networks," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 70, PMLR, 2017, pp. 136–145.

[89] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and Z. Kolter, "Differentiable convex optimization layers," in *Advances in Neural Information Processing Systems*, 2019.

[90] B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable mpc for end-to-end planning and control," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31, Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/ba6d843eb4251a4526ce65d1807a9309-Paper.pdf.

[91] P.-W. Wang, P. L. Donti, B. Wilder, and J. Z. Kolter, "Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver," in *International Conference on Machine Learning*, 2019.

[92] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[93] J. Wright and Y. Ma, *High-Dimensional Data Analysis with Low-Dimensional Models: Principles, Computation, and Applications*. Cambridge University Press, 2021.

[94] A. Beck and M. Teboulle, "Mirror descent and nonlinear projected subgradient methods for convex optimization," *Operations Research Letters*, vol. 31, no. 3, pp. 167–175, 2003, ISSN: 0167-6377. DOI: https://doi.org/10.1016/S0167-6377(02)00231-6. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167637702002316.

[95] C. Blair, "Problem complexity and method efficiency in optimization (a. s. nemirovsky and d. b. yudin)," *SIAM Review*, vol. 27, no. 2, pp. 264–265, 1985. DOI: 10.1137/1027074. eprint: https://doi.org/10.1137/1027074. [Online]. Available: https://doi.org/10.1137/1027074.

[96] B. Amos, L. Xu, and J. Z. Kolter, "Input convex neural networks," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 70, PMLR, 2017, pp. 146–155.

[97] K. B. Petersen and M. S. Pedersen, *The matrix cookbook*, Version 20081110, Oct. 2008. [Online]. Available: http://www2.imm.dtu.dk/pubdb/p.php?3274.

[98] G. Garrigos and R. M. Gower, "Handbook of convergence theorems for (stochastic) gradient methods," *arXiv preprint arXiv:2301.11235*, 2023.

[99] A. Razavi, A. van den Oord, and O. Vinyals, "Generating diverse high-fidelity images with vq-vae-2," in *Neural Information Processing Systems*, 2019.

[100] P. Esser, R. Rombach, and B. Ommer, *Taming transformers for high-resolution image synthesis*, 2020. arXiv: 2012.09841 [cs.CV].

[101] A. van den Oord, O. Vinyals, and K. Kavukcuoglu, "Neural discrete representation learning," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6309–6318, ISBN: 9781510860964.

[102] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, *Jukebox: A generative model for music*, 2020. arXiv: 2005.00341 [eess.AS].

[103] X. Chen, N. Mishra, M. Rohaninejad, and P. Abbeel, *Pixelsnail: An improved autoregressive generative model*, 2017. arXiv: 1712.09763 [cs.LG].

[104] S. Gu, D. Chen, J. Bao, F. Wen, B. Zhang, D. Chen, L. Yuan, and B. Guo, "Vector quantized diffusion model for text-to-image synthesis," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 10 686–10 696. DOI: 10.1109/CVPR52688.2022.01043.

[105] G. Wallace, "The jpeg still picture compression standard," *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992. DOI: 10.1109/30.125072.

[106] D. P. Kingma, D. J. Rezende, S. Mohamed, and M. Welling, "Semi-supervised learning with deep generative models," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'14, Montreal, Canada: MIT Press, 2014, pp. 3581–3589.

[107] D. P. Kingma and M. Welling, "An introduction to variational autoencoders," *Foundations and Trends® in Machine Learning*, vol. 12, no. 4, pp. 307–392, 2019, ISSN: 1935-8237. DOI: 10.1561/2200000056. [Online]. Available: http://dx.doi.org/10.1561/2200000056.

[108] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, "Zero-shot text-to-image generation," in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., ser. Proceedings of Machine Learning Research, vol. 139, PMLR, Jul. 2021, pp. 8821–8831. [Online]. Available: https://proceedings.mlr.press/v139/ramesh21a.html.

[109] Y. Bengio, N. Léonard, and A. C. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *ArXiv*, vol. abs/1308.3432, 2013.

[110] L. Kaiser, S. Bengio, A. Roy, A. Vaswani, N. Parmar, J. Uszkoreit, and N. Shazeer, "Fast decoding in sequence models using discrete latent variables," in *Proceedings of the 35th International Conference on Machine Learning*, J. Dy and A. Krause, Eds., ser. Proceedings of Machine Learning Research, vol. 80, PMLR, Jul. 2018, pp. 2390–2399. [Online]. Available: https://proceedings.mlr.press/v80/kaiser18a.html.

[111] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," in *International Conference on Learning Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=rkE3y85ee.

[112] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," in *International Conference on Learning Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=S1jE5L5gl.

[113] N. Zeghidour, A. Luebs, A. Omran, J. Skoglund, and M. Tagliasacchi, "Soundstream: An end-to-end neural audio codec," *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, vol. 30, pp. 495–507, Nov. 2021, ISSN: 2329-9290. DOI: 10.1109/TASLP.2021.3129994. [Online]. Available: https://doi.org/10.1109/TASLP.2021.3129994.

[114] M. Huh, B. Cheung, P. Agrawal, and P. Isola, "Straightening out the straight-through estimator: Overcoming optimization challenges in vector quantized networks," in *International Conference on Machine Learning*, PMLR, 2023.

[115] D. Lee, C. Kim, S. Kim, M. Cho, and W.-S. Han, "Autoregressive image generation using residual quantization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 11 523–11 532.

[116] P. Yin, J. Lyu, S. Zhang, S. J. Osher, Y. Qi, and J. Xin, "Understanding straight-through estimator in training activation quantized neural nets," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=Skh4jRcKQ.

[117] Y. Takida, T. Shibuya, W. Liao, *et al.*, "SQ-VAE: Variational bayes on discrete representation with self-annealed stochastic quantization," in *International Conference on Machine Learning*, 2022.

[118] T. Gautam, R. Pryzant, Z. Yang, C. Zhu, and S. Sojoudi, "Soft convex quantization: Revisiting vector quantization with convex optimization," *Technical report*, 2023. [Online]. Available: https://arxiv.org/pdf/2310.03004.pdf.

[119] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 586–595. DOI: 10.1109/CVPR.2018.00068.