UNIVERSITY OF CALIFORNIA

Los Angeles

Fast Training of Generalizable Deep Neural Networks

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Electrical and Computer Engineering

by

Omead Brandon Pooladzandi

2023

ABSTRACT OF THE DISSERTATION

Fast Training of Generalizable Deep Neural Networks

by

Omead Brandon Pooladzandi
Doctor of Philosophy in Electrical and Computer Engineering
University of California, Los Angeles, 2023
Professor Gregory J. Pottie, Chair

Effective natural agents excel in learning representations of our world and efficiently generalizing to make decisions. Critically, developing such advanced *reasoning* capabilities can occur even with *limited information-rich samples*. In stark contrast, the major success of deep learning-based artificial agents is primarily trained on massive datasets. This dissertation focuses on curvature-informed learning and generative modeling methods that boost efficiency and close the gap between natural and artificial agents, thus enabling computationally efficient and improved reasoning.

This dissertation is comprised of two parts. First, we formally lay the foundations for *learning*. The goal is to establish optimization techniques, understand datasets, establish probabilistic generative models, and provide natural learning objectives even in settings with limited supervision. We discuss various first and second-order optimization methods, show the importance of modeling distributions in Variational Auto Encoders (VAEs), and discuss which points are essential for generalization in supervised learning. Building on these insights, we develop new algorithms to boost the performance of state-of-the-art models, select subsets to improve data quality, speed up training, mitigate their biases, and generate new augmentations on large labeled and partially labeled datasets. These contributions enable

ML systems to better model and generalize to unseen and potentially out-of-distribution samples while drastically reducing training time and computational cost.

The dissertation of Omead Brandon Pooladzandi is approved.

Lieven Vandenberghe

Jonathan Kao

Baharan Mirzasoleiman

Zaid Towfic

Gregory J. Pottie, Committee Chair

University of California, Los Angeles

2023

To my parents.

Contents

List of Figures

List of Tables

Preface

I would like to thank my Ph.D. advisor, Gregory Pottie, for being the support I needed and giving me the freedom to explore all these topics. To my dear lab mates Sunay Bhat and Jeffery Jiang, whom I worked closely with on many projects and who have been kind, supportive, and always excited to brainstorm and try out crazy new ideas together. My time in Professor Pottie's lab has been a blessing.

My experience at UCLA would be incomplete without the company of my dear friends Pasha and Lev, who were always eager to chat about research or play games. Arash and Darius for always having an open door and exploring LA with me. Brandon, Brayan, Xavier, and the rest of Flush for coaxing me to take time away from school and relax since undergrad. Calvin, for keeping me sane through the pandemic. And Joseph for being my best friend since high school. Thank you, Tara, for always being there for me no matter the situation; you provide me with stability and reason.

I have also had the privilege of standing on the shoulders of amazing mentors and collaborators over the years: Barbra Hudgins, who taught me calculus and believed in me in high school; Glenn Healey, who shared his love of classical computer vision during undergrad with me, Zaid Towfic, who pushed me at JPL and UCLA, Jonathan Kao, who taught me the fundamentals of Deep Learning, Lieven Vandenberghe, who taught me convex optimization, Baharan Mirzasolaiman, who taught me how to write my first research paper and to optimize rebuttals, and Xi-Lin Li who taught me the beauty of Lie groups. I am also grateful to all my additional collaborators and administrators who kept the wheels turning.

And to Ryo Arreola and Deeona Columbia, who so kindly helped me navigate the treacherous waters of my master's and Ph.D. For without their guidance, this work would not be.

Finally, and certainly above all, I will forever remain indebted to my parents, Arezou and Shahbahram, for their unconditional love, unwavering encouragement, boundless support, and steadfast care for my happiness and well-being. This one is for you.

Curriculum Vitae

2014 – 2017	B.S. in Electrical Engineering, University of California, Irvine (UCI), Irvine, California.
2017 – 2018	M.S. in Electrical and Computer Engineering, University of California, Los Angeles (UCLA), Los Angeles, California.
2017 – 2023	Graduate Research Assistant at The University of California, Los Angeles (UCLA), Los Angeles, California.
2023	Teaching Assitant at The University of California, Los Angeles (UCLA), Los Angeles, California.

REFEREED CONFERENCE PUBLICATIONS

Adaptive Second Order Coresets for Data-efficient Machine Learning **2022**
International Conference on Machine Learning

Generating High Fidelity Synthetic Data via Corset selection and Entropic Regularization **2022**
Neural Information Processing Systems

Improving Levenberg-Marquardt Algorithm for Neural Networks **2022**
Neural Information Processing Systems

Causal Structural Hypothesis Testing and Data Generation Models **2022**
Neural Information Processing Systems

Towards Mapping Latent Space Representations via Image Augmentation Priors **2023**
Information Theory Applications

De-Biasing Generative Models using Counterfactual Methods **2022**
Information Theory Applications

Exploring the Potential of VAE Decoders for Enhanced Speech Re-Synthesis **2023**
IEEE Statistical Signal Processing

REFEREED PREPRINTS

Curvature-Informed SGD via General Purpose Lie-Group Preconditioners **2023**
Neural Information Processing Systems

CHAPTER 1

Introduction

Learning amounts to elucidating the unknown. The ability to effectively combine information and make accurate inferences or generalizations has been a long-standing goal of Artificial-intelligence (AI) research. In the last few decades, there has been an explosion in the scale of datasets, algorithmic complexity, and computational resources available to allow machine learning models to capture complex data with high fidelity. We have witnessed the emergence of Generative Adversarial Networks (GANs), Variational Auto Encoders (VAEs), and diffusion models, along with a profusion of novel optimizers, transformers, and deep causal modeling techniques. These new massive architectures, datasets, and computationally intensive methods provide ample room for industry to make huge leaps forward like chatGPT-4 and Stable Diffusion, both incredibly popular at the time of writing this dissertation. Yet, many fundamental questions about learning remain unanswered. This dissertation focuses on methods for identifying, selecting, and generating the most salient representations of large datasets, discovering new learning dynamics facilitated via optimization, manipulating latent representations of data, and exploring the effect of generative modeling under different distribution assumptions. All these techniques are to enable a deeper form of learning for state-of-the-art AI.

1.1 Approach

To improve machine learning we need first to understand what makes it work. Three of the most important factors of machine learning are 1) Rich Datasets 2) Powerful, Efficient, and

Generalizable Optimizers, and 3) Distributional Modeling. If we are armed with these three approaches we can model the unknown.

In the field of machine learning, rich and comprehensive datasets play a critical role in enabling algorithms to learn and make accurate predictions. The abundance of data provides a foundation on which models can be trained to recognize complex patterns and make informed decisions. As such, the quality of the dataset is a key determinant of the performance of a machine-learning model.

One of the key advantages of rich datasets is the ability to improve the generalization capabilities of the model. By providing a diverse range of examples, a dataset enables a model to learn patterns that are representative of the true underlying distribution of data. This, in turn, allows the model to make accurate predictions on new, previously unseen data. On the other hand, insufficient or biased datasets can lead to models that fail to generalize and perform poorly on new tasks.

To motivate the efficient *identification of the most salient points of a dataset*, consider the recent study by Toneva (TSC18) which showed more than half of the data points of many large-scale datasets can be pruned without loss in generalization. This shows that certain points are crucial to learning. Hence if these defining points can be efficiently identified, selected, and generated we should see a reduction in modeling the unknown, while reducing the substantial computational costs of training machine learning models on massive datasets. To alleviate such costs, there has been a sustained effort (MBL19; BMB19; KF18; TSC18; MBL20) to develop data-efficient training methods that can carefully select subsets of training data that generalize on par with the full dataset. Further, generative models can synthesize data points drawn from the data distribution, however, not all generated samples are high quality. Naively training on all points drawn from a generative model will increase uncertainty in the overall learning system. As such to reduce uncertainty in the system, one needs to generate high-fidelity points while filtering out bad-quality points. Note, what serves as high-quality information for one training regime might be bad for another.

Stochastic optimization is the workhorse of modern Deep Learning as it allows for efficient and effective optimization of complex models with large datasets. By randomly selecting subsets of the data, stochastic optimization algorithms such as stochastic gradient descent and Adam (KB14) can converge faster and avoid getting stuck in local minima. This makes it possible to train deep neural networks with millions of parameters, which would be infeasible with traditional optimization methods. First-order optimization methods such as SGD & Adam have driven DL forward by finding generalizable solutions that minimize risk. This ultimately allows us to model and draw inferences from data distributions using DL. Very recently, Li (Li18b) and Martines (MG15b) showed that second-order optimizers can be applied to deep learning methods to outperform first-order methods. Yet in practice, most applications still use first-order methods due to scaling issues. Forming an efficient, scalable, and generalizable higher-order optimizer can potentially improve DL.

Distributional modeling is a critical aspect of deep learning that has become increasingly relevant in recent years. Deep learning models are often trained on large datasets, which can exhibit complex and varied distributions. To capture the patterns within these datasets, it is essential to model the distribution of the data accurately. By doing so, we can gain insights into the underlying structure of the data, which can help us to develop more robust and effective models. Currently, most traditional deep-learning models assume that the data is normally distributed, which may not always be the case. By incorporating more flexible distributional models, it is possible to better capture the underlying structure and true prior distribution of the data to improve performance. Once we have learned a distribution of data, generating data from the model is equivalent to sampling from this induced distribution. This leads to an entire sub-field of deep learning called generative modeling.

1.2 General Contribution

Throughout my Ph.D., I have had the opportunity to research many exciting topics. These have fallen under two nonexclusive thematic groups: curvature-informed deep learning and generative modeling. One crucial insight I received through Professor Vandenberghe's optimization class is that preconditioning with curvature information can greatly improve a gradient's quality. I first surveyed second-order methods and implemented an improved version of the Levenberg–Marquardt method presented at HOO NeurIPs 2022 (PZ22). Using the insight from this paper, I began working on a subset selection method named AdaCore that used submodular optimization to select an optimal subset whose curvature preconditioned gradients covered the preconditioned gradients of the entire dataset. This resulted in a significant speedup and accuracy of deep learning models, published at ICML 2021 (PDM22a). Finally, during my internship with Meta, I developed two families of efficient general-purpose curvature-informed gradient preconditioners. Although in this dissertation we mostly discuss preconditioning Stochastic Gradient Descent, the preconditioners can be used for any task.

With respect to generative modeling, I have worked on pairing Latent Energy-Based Models (LEBMs) with coreset methods to guide the generative model to automatically learn the best augmentation distribution for a given dataset via semi- and self-supervised learning (PKN23). Furthermore, I have worked on generative causal modeling, where one uses prior causal knowledge to restrict, manipulate and map the latent space of a VAE to generate out-of-distribution samples (PJB23; BJP22; JPB22). Finally, during my internship with Meta, I studied the effect of a VAEs decoder distribution for speech re-synthesis. This work, published at IEEE SSP (PLG23), found that the gamma distribution is particularly well-suited for speech re-synthesis.

1.3 Dissertation Overview

This dissertation will concentrate primarily on the research papers that I have published. Specifically, I will describe an efficient second-order optimizer, Preconditioned Stochastic Gradient Descent (PSGD), which exploits curvature information for improving convergence rates. Additionally, I will discuss Coreset selection for speeding up training, which leverages curvature information and provides provable convergence guarantees (AdaCore). Furthermore, I will describe my research on Latent Energy-Based Models (L-EBM), which utilizes Coresets and Entropy Regularization to enhance classification generalization and generation quality for contrastive and semi-supervised learning. Finally, I will investigate different decoder distributions to improve speech re-synthesis (Gamma-VAE).

This dissertation will discuss:

- **Chapter 2 - Background :** we provide the necessary background to machine learning and briefly review some key prior works.
- **Chaper 3 - Preconditioned Stochastic Gradient Descent (PSGD):** a second-order optimization method that outperforms state-of-the-art first and second-order optimization methods, such as SGD, AdaBelief (ZTD20), KFAC (MG15a), and Shampoo (GKS18), with only negligible computational overhead compared to SGD. Moreover, PSGD demonstrates better neuroplasticity than other optimizers.
- **Chapter 4 - AdaCore:** a theoretically rigorous subset selection method that extracts the most salient subsets from large datasets, significantly reducing the computational costs associated with training models on massive datasets. This is accomplished by selecting a subset whose curvature preconditioned gradient matches that of the full dataset.
- **Chapter 5 - LEBM + Coresets:** Enhanced performance of contrastive models in

semi- and self-supervised settings – focusing on low entropy and forgettable points selected through coreset methods from a generative model.

- **Chapter 6 - Gamma-VAE:** exploring the potential of different Variational Autoencoder (VAE) decoders for improved speech resynthesis.
- **Chapter 7 - Conclusion:** summarizing what we have learned and what we can do in the future, as well as some very preliminary experiments showing direct future directions of this work.

CHAPTER 2

Fundamental Background

This dissertation covers many general topics in deep learning. To provide some background we will review some core concepts needed for this dissertation. We will begin by discussing some basic math from optimization needed for deep learning. Then we will discuss some simple learning regimes such as classification and generation.

Next, we will go over some recent work by (TSC18) which elucidates which points of the dataset are important for generalization in the supervised setting. This will set the scene for understanding points selected for coreset selection for supervised and self- and semi-supervised learning well as the behavior of optimizers.

Finally, we will review some basics about the Variational Auto Encoder (VAE) which will be used as the base generative model for Chapter 5 and Chapter 6.

2.1 Empirical Risk Minimization

Training machine learning models often reduces to minimizing an empirical risk function. Given a not-necessarily convex loss $\mathcal{L}$, one aims to find the model parameter vector θ_* over the parameter space Θ that minimizes the loss $\mathcal{L}$ over the training data $\mathcal{X}$:

$$\theta_* \in \arg\min_{\theta \in \Theta} \mathcal{L}(\theta), \tag{2.1}$$
$$\mathcal{L}(\theta) := \sum_{i \in \mathcal{X}} \ell_i(\theta), \quad \ell_i(\theta) = \ell(f(x_i, \theta), y_i),$$

where $\mathcal{X} = \{1, \ldots, n\}$ is an index set of the training data, $\theta \in \mathbb{R}^d$ is the parameter set of the model f being trained, and ℓ_i is the loss function associated with training example $i \in \mathcal{X}$ with feature vector $x_i \in \mathbb{R}^d$ and label y_i.

In binary classification, $\mathcal{Y} = 0, 1$, and a commonly used loss function is the logistic loss:

$$\ell(\hat{y}, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}), \tag{2.2}$$

where $y \in 0, 1$ is the true label, and $\hat{y} \in [0, 1]$ is the predicted probability of the positive class. The logistic loss is a convex surrogate for 0-1 loss, which is the loss function that measures the error of the classifier in terms of the misclassification rate. However, 0-1 loss is non-convex and difficult to optimize, hence the use of logistic loss as a proxy.

In multiclass classification, $\mathcal{Y}$ contains more than two classes, and a commonly used loss function is the cross-entropy loss:

$$\ell(\hat{y}, y) = -\sum_{k=1}^{K} y_k \log(\hat{y}_k), \tag{2.3}$$

where $y \in 0, 1^K$ is a one-hot encoding of the true label, and $\hat{y} \in [0, 1]^K$ is the predicted probability vector over the K classes. The cross-entropy loss measures the negative log-likelihood of the true label under the predicted probability distribution. The performance of a classification model is often evaluated on a held-out test set using metrics such as accuracy, precision, recall, F1 score, or area under the receiver operating characteristic (ROC) curve. These metrics quantify the performance of the classifier in terms of its ability to correctly predict the true class labels.

2.1.1 Unsupervised learning

Unsupervised learning is a type of learning where the goal is to learn the underlying structure of the data without using labeled training data. Formally, given an unlabeled dataset $x_{ii=1}^{n}$, where $x_i \in \mathcal{X}$ is input, the goal of unsupervised learning is to learn a function $f : \mathcal{X} \to \mathcal{Z}$ that maps each input to a lower-dimensional representation $\mathbf{z} \in \mathcal{Z}$. Unsupervised learning algorithms typically involve the optimization of some objective function that captures the structure of the data, such as clustering, dimensionality reduction, or generative modeling. The performance of unsupervised learning algorithms is often evaluated using measures such as clustering accuracy, reconstruction error, or likelihood of the generative model.

2.1.2 Semi-supervised learning

Semi-supervised learning is a type of learning where both labeled and unlabeled data are used to learn a model. Formally, given a dataset $(x_i, y_i)i = 1^n$, where $x_i \in \mathcal{X}$ is the input and $y_i \in \mathcal{Y}$ is the corresponding label for the labeled data, and an unlabeled dataset $x_{ii=n+1}^{N}$, where N is the total number of examples, the goal of semi-supervised learning is to learn a function $f : \mathcal{X} \to \mathcal{Y}$ that minimizes the expected risk over both labeled and unlabeled data:

$$R(f) = \mathbb{E}_{(x,y)\sim p(x,y)} \left[\ell(f(x), y)\right], \tag{2.4}$$

where ℓ is the loss function as in supervised learning, and the expectation is taken over both labeled and unlabeled data. Semi-supervised learning algorithms typically involve a combination of supervised and unsupervised learning, where the objective function is optimized over both labeled and unlabeled data. The performance of semi-supervised learning algorithms is evaluated using metrics such as accuracy on the labeled data or classification performance on the unlabeled data.

2.1.3 Generation

Generative modeling is a type of unsupervised learning problem that involves learning the underlying probability distribution of the data. The goal is to learn a model $p_{\mathrm{g}}(x)$ that can generate new samples from the same distribution as the training data.

Formally, given a training dataset $(x_i)_{i=1}^n$, where $x_i \in \mathcal{X}$ and $\mathcal{X}$ is the data space, the goal of generative modeling is to find a generative model $p_{\mathrm{g}}(x)$ that approximates the true data distribution $p_{\mathrm{data}}(x)$. This can be done by maximizing the log-likelihood of the training data under the generative model:

$$\ell_{\mathrm{train}}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \log p_{\mathrm{g}}(x_i); \theta), \tag{2.5}$$

where θ is the parameter set of the generative model.

2.1.3.1 Variational Auto Encoders

The Variational Auto Encoder (VAE) (KW13a) is a framework that learns a latent representation, z, of the data x, which captures the underlying structure of the data distribution. VAEs are trained to learn an encoder function, $e_\phi(z|x)$, which maps the input data x to the latent space z. The approximate posterior distribution over the latent variables is given by $e_\phi(z|x)$. Typically, the encoder estimates the parameters of a normal distribution in the latent space, such as the mean and variance, which define the location and scale of the distribution. Recently, (TIY18) showed that encoding the latent variables as a student-t distribution can lead to more stable training of a VAE as well as enable the generator to produce higher-quality samples. The VAE also learns a marginalized distribution, $m_\theta(z)$, for the latent variable encoding, and a decoder function, $d_\theta(x|z)$. Thus the joint distribution, $p_\theta(x, z)$, can accurately model the true data distribution, $p(x)$. The decoder estimates the parameters of a distribution, such as the shape and rate parameters of a Gamma distribution, to reconstruct the input signal x. The decoding side defines a joint pdf, $q_\theta(x, z) = m_\theta(z)d_\theta(x|z)$, with everything explicit, resulting in a joint pdf, $p_\phi(x, z) = p(x)e_\phi(z|x)$ that we can readily draw samples from, although $p(x)$ is unknown.

The VAE is based on the principle of maximum likelihood estimation (MLE), where the goal is to maximize the likelihood of the training data under the model. To make the

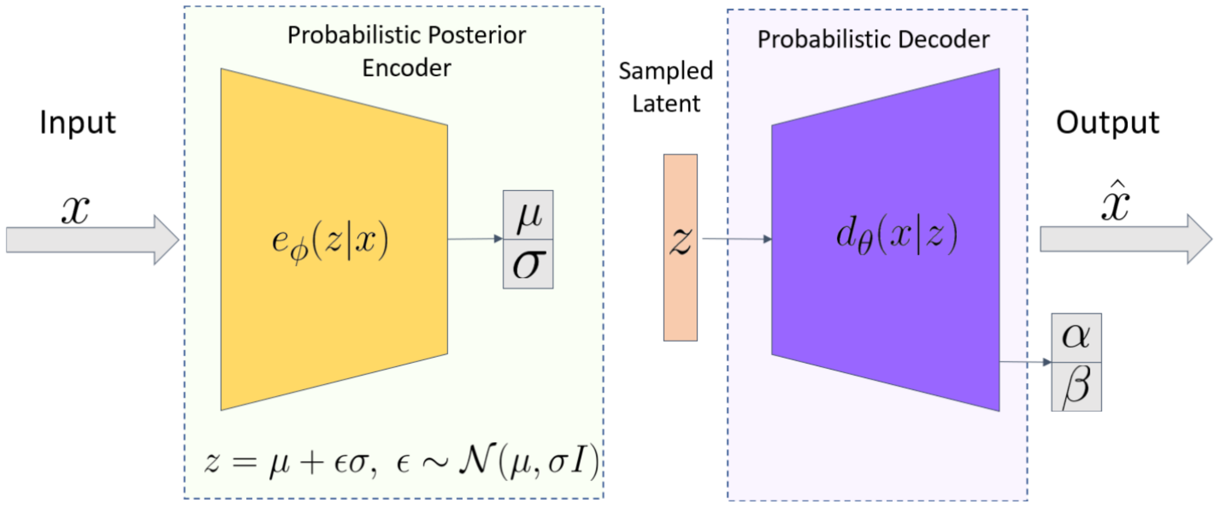


Figure 2.1: VAE: Where x is the input signal and $\hat{x}$ is estimated via the parameters of the decoder distribution.

optimization process more feasible, the VAE introduces the variational reparameterization trick, which involves re-parameterizing the random noise used to sample from the latent distribution so that the resulting samples can be transformed into the latent space in a differentiable manner. By using this trick, VAEs can be trained more efficiently on large datasets using a variational lower bound on the log-likelihood, written as:

$$\mathcal{L}(\theta, \phi; x) = \mathbb{E}_{e_\phi(z|x)} \left[\log d_\theta(x|z)\right] - \mathrm{KL}\left(e_\phi(z|x) || m_\theta(z)\right) \tag{2.6}$$

where x is a single training example, and θ and ϕ are the parameters of the decoder and encoder, respectively. The first term in the lower bound, $\mathbb{E}_{e_\phi(z|x)} \left[\log d_\theta(x|z)\right]$, is known as the reconstruction loss, and it measures the difference between the reconstructed data and the original data. The second term, $\mathrm{KL}\left(e_\phi(z|x) || m_\theta(z)\right)$, is known as the KL divergence, and it measures the difference between the approximate posterior distribution and the latent distribution.

2.1.4 Optimization

To be able to train any of the above types of machine learning models we need some sort of optimization mechanism to minimize a given loss function. We denote the gradient of the loss w.r.t. model parameters by $\mathbf{g} = \nabla\mathcal{L}(\theta) = \frac{1}{|\mathcal{X}|}\sum_{i\in V}\frac{\partial \ell_i}{\partial \theta}$, and the corresponding second derivative (i.e., Hessian) by $\mathbf{H} = \nabla^2\mathcal{L}(\theta) = \frac{1}{|\mathcal{X}|}\sum_{i\in\mathcal{X}}\frac{\partial^2 \ell_i}{\partial\theta_j\partial\theta_k}$.

First-order gradient methods are popular for solving Problem (2.1). Such methods start from an initial point θ_0, and every iteration t steps in the negative direction of the gradient. The most popular first-order method, Stochastic Gradient Descent (SGD) (RM51), is often used with momentum accelerating it in directions whose gradients point in the same directions and dampens oscillations in dimensions whose gradients change directions (Qia99):

$$\theta_{t+1} = \theta_t - \eta_t\mathbf{v}_t, \qquad \mathbf{v}_t = \mathbf{g}_t, \tag{2.7}$$

Here, η_t is the learning rate, and $\mathbf{g}_t$ is the gradient of a batch at the t-th iteration. SGD is, however, sensitive to learning rate, as it uses the same learning rate for different dimensions with possibly very different scales. To address this, adaptive first-order methods such as AdaGrad (DHS11), RMSProp (HSS12), AdaDelta (Zei12), and Adam (KB17) perform smaller updates (i.e. smaller learning rates) for parameters that frequently have a large gradient, and larger updates (i.e. larger learning rates) for parameters that rarely have a large gradient.

Second-order gradient methods rely on the geometry of the problem to automatically rotate and scale the gradient vectors, using the curvature of the loss landscape. In doing so, second-order methods can both choose a better descent direction and automatically adjust the learning rate for each parameter. Hence, second-order methods have superior convergence properties compared to first-order methods. Newton's method (Ber82) is a classical second-order method that preconditions the gradient vector with the inverse of the local Hessian at every iteration, $\mathbf{H}_t^{-1}$:

$$\theta_{t+1} = \theta_t - \eta_t \mathbf{H}_t^{-1} \mathbf{g}_t. \tag{2.8}$$

As inverting the Hessian matrix requires quadratic memory and cubic computational complexity, several methods that approximate the Hessian information significantly reduce time and memory complexity (BLN95; SZL13; MG15b; Li15; Li19; XRM20). In addition, EGD (DVB15) and later AdaHessian (YGS20) directly approximates the diagonal of the Hessian parameter update.

2.1.4.1 Quasi-Newton Method

The quasi-Newton method uses inverse Hessian estimation to compute each update iteration. The most popular one is the Broyden–Fletcher–Goldfarb–Shanno (BFGS) update.

$$H_{k+1}^{-1} = (I - \frac{sy^T}{y^T s}) H_k^{-1} (I - \frac{ys^T}{y^T s}) + \frac{ss^T}{y^T s} \tag{2.9}$$

$$s = x_{k+1} - x_k \quad y = \nabla f(x_{k+1}) - \nabla f(x_k) \tag{2.10}$$

where H_k^{-1} is the approximation of inverse Hessian at iteration k. Then to further tackle the memory issue, limited-memory BFGS (L-BFGS) was proposed to avoid storing H_k and H_k^{-1} by evaluating H_k recursively for m iterations ($j = k-1, ..., k-m$).

$$H_{j+1}^{-1} = (I - \frac{s_j y_j^T}{y_j^T s_j}) H_j^{-1} (I - \frac{y_j s_j^T}{y_j^T s_j}) + \frac{s_j s_j^T}{y_j^T s_j} \tag{2.11}$$

$$s_j = x_{j+1} - x_j \quad y_j = \nabla f(x_{j+1}) - \nabla f(x_j) \tag{2.12}$$

Very recently, this low-rank curvature-aware optimization method has been adjusted for use in deep learning optimization (GRB20).

2.1.4.2 Hessian-Free

Hessian-free optimization uses the insights from Newton's method but comes up with a more efficent way to minimize the quadratic function. Just as in Newton's method, given a function f, we find the second-order Taylor expansion:

$$f(x+v) \approx f(x) + \nabla f(x)^T v + v^T H(f) v \tag{2.13}$$

We then find the minimum of this approximation (the best Δx) using the conjugate gradient method, take an iterative step to $x = x + v$, and continue this iteration until we have convergence.

Algorithm 1 Hessian-Free

Let f be any function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ which we wish to minimize.

1. Initialize:
   Let $i = 0$ and $x_i = x_0$ be some initial guess.

2. Set up quadratic expansion:
   At the current x_n, compute the gradient $\nabla f\,(x_n)$ and Hessian $H(f)\,(x_n)$, and consider the following Taylor expansion of f :

   $$f(x+v) \approx f(x) + \nabla f(x)^T v + v^T H(f) v$$

3. Conjugate gradient:
   Compute x_{n+1} using the Conjugate Gradient algorithm for quadratic functions on the current Taylor expansion.

4. Iterate:
   Repeat steps 2 and 3 until x_n has converged.

Fortunately in the Hessian free algorithm, it is possible to circumvent the full Hessian expansion in step 2, and instead only requires us to calculate the much cheaper Hessian vector product Hv.

To see this, consider the first component of Hv. The $i-$th row of the Hessian contains partial derivatives of the form $\frac{\partial^2}{\partial x_i x_j} f$ where j is the column index. As per normal matrix-vector multiplication, the $i-$th row of Hv is the dot product of v and ith row of H. The $i-$th row of H can be expressed as the gradient of the derivative of f with respect to x_i, yielding (Gib14):

$$(Hv)_i = \sum_{j=1}^{N} \frac{\partial^2 f}{\partial x_i x_j}(x) \cdot v_j = \nabla \frac{\partial f}{\partial x_i}(x) \cdot v$$

One can utilize auto-differentiation packages to calculate the exact Hessian vector product as detailed by (Pea94) The operator is defined as

$$\mathcal{R}_{\mathbf{v}}\{f(\mathbf{x})\} \equiv \frac{\partial}{\partial t} f(\mathbf{x} + t\mathbf{v})\bigg|_{t=0}$$

so $\mathbf{Hv} = \mathcal{R}_{\mathbf{v}}\left\{\nabla_{\mathbf{x}}(\mathbf{x})\right\}$. (To avoid clutter we will usually write $\mathcal{R}\{\cdot\}$ instead of $\mathcal{R}_{\mathbf{v}}\{\cdot\}$) We can then take all the equations of a procedure that calculates a gradient, *e.g.* the backpropagation procedure, and we can apply the $\mathcal{R}_v\{\cdot\}$ operator to each equation. Because $\mathcal{R}\{\cdot\}$ is a differential operator, it obeys the usual rules for differential operators, such as:

$$\begin{aligned}
\mathcal{R}\{cf(\mathbf{x})\} &= c\mathcal{R}\{f(\mathbf{x})\} \\
\mathcal{R}\{f(\mathbf{x}) + g(\mathbf{x})\} &= \mathcal{R}\{f(\mathbf{x})\} + \mathcal{R}\{g(\mathbf{x})\} \\
\mathcal{R}\{f(\mathbf{x})g(\mathbf{x})\} &= \mathcal{R}\{f(\mathbf{x})\}g(\mathbf{x}) + f(\mathbf{x})\mathcal{R}\{g(\mathbf{x})\} \\
\mathcal{R}\{f(g(\mathbf{x}))\} &= f'(g(\mathbf{x}))\mathcal{R}\{g(\mathbf{x})\} \\
\mathcal{R}\left\{\frac{df(\mathbf{x})}{dt}\right\} &= \frac{d\mathcal{R}\{f(\mathbf{x})\}}{dt}
\end{aligned}$$

Also, note that

$$\mathcal{R}\{\mathbf{x}\} = \mathbf{v}$$

These rules allow one to use existing auto-differentiation packages to calculate the vector $\mathcal{R}\{\nabla_{\mathbf{x}}\}$ which is precisely the vector which we desire Hv.

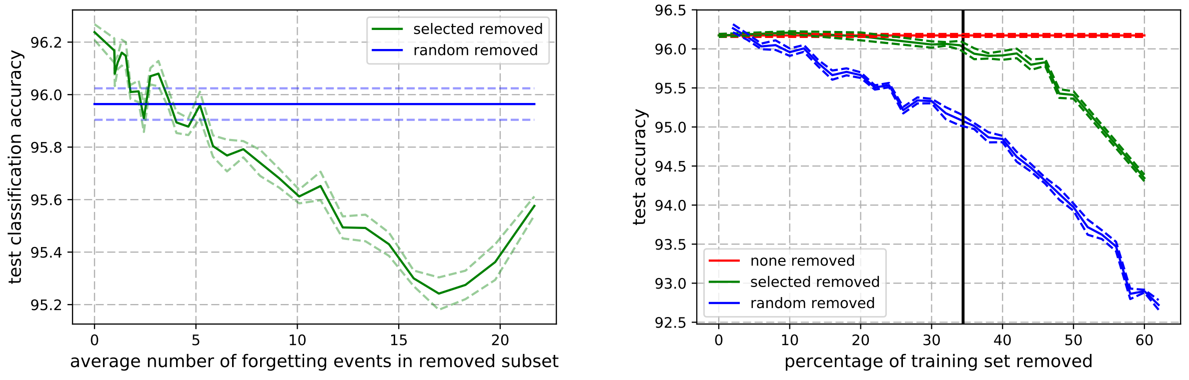
In practice, HF sees fewer applications than SGD because its updates are much more expensive to compute, as they involve running linear conjugate gradient (CG) for potentially hundreds of iterations. Each of these iterations requires a matrix-vector product with the curvature matrix (which is as expensive to compute as the stochastic gradient on the current mini-batch). Next, HF's estimate of the curvature matrix must remain fixed while CG iterates. This means that the method is able to go through much less data than SGD can in a comparable amount of time, making it less well-suited to stochastic optimizations.

2.1.5 Learning

When an entity learns, whether it be a human or a machine learning algorithm, it is natural for certain concepts or definitions to be learned and forgotten. Sometimes all it takes for an entity to learn is a single presentation of the information, whereas for others it may take multiple presentations. The concept of forgetting is defined in machine learning as data points that are learned or classified correctly at some point during training, and then at a later time forgotten or classified incorrectly. Datapoints that are learned and then forgotten during training are called unforgettable points. Points that are learned, and throughout training are not forgotten are considered forgettable points. The event for which a point that has been learned or is being classified correctly, is forgotten or classified incorrectly, is called a forgetting event.

As discussed in (TSC19a), the procedure to collect forgetting statistics is computationally expensive as one needs to train a network until completion to observe the forgetting statistics. Instead, they offer the insight that the unforgettable points in a dataset can be removed from training without a decrease in accuracy compared to the full dataset. An insight that can be drawn from the nature of data points in machine learning datasets from this work is that a considerable speedup in training will occur if one can quickly remove the unforgettable points during training.

Removing Forgettable & Unforgettable Datapoints A simple way to see the importance of forgettable points is by removing them from a training set and seeing the decrease in accuracy. Furthermore, the unimportance of unforgettable points is shown by removing them from the training set and seeing that the accuracy does not drop. These scenarios are shown in Fig. 2.2.

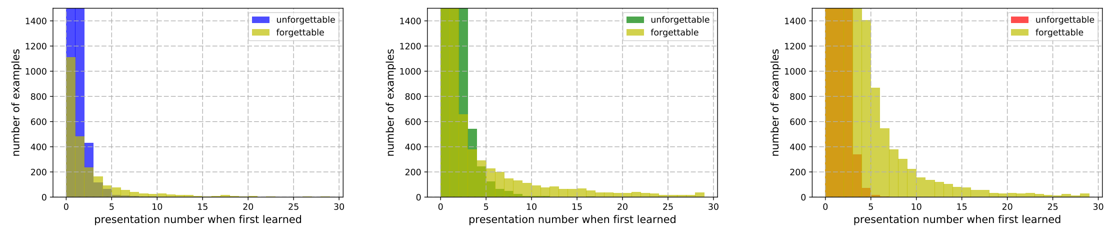


(a) Difference in generalization performance (b) Resnet18/CIFAR-10: Generalization performance

Figure 2.2: Left Difference in generalization performance when contiguous chunks of 5000 increasingly forgotten examples are removed from the training set. The most important examples tend to be those that are forgotten the most. Right Generalization performance on CIFAR-10 of ResNet18 where increasingly larger subsets of the training set are removed (mean +/- std error of 5 seeds). When the removed examples are selected at random, performance drops very fast. Selecting the examples according to forgetting ordering can reduce the training set significantly without affecting generalization. The vertical line indicates the point at which all unforgettable examples are removed from the training set.

Presentations to Learn Once one has separated a dataset into forgettable and unforgettable points, an interesting statistic is to see the distribution of presentations required to learn points in each group. In Fig 2.3 we see that the seemingly unimportant unforgettable points are mostly learned within the first 3-5 presentations depending on the dataset. In contrast, the distribution of presentations to learn the unforgettable points is long-tailed and may take more than 20 epochs to learn. Note that the unforgettable "easy" data points may have a low loss during most of the training.

Entropy and Margin Entropy and Margin are two metrics that can be used to identify



(a) MNIST (b) permutedMNIST (c) CIFAR-10

Figure 2.3: Distributions of the first presentation at which each unforgettable and forgettable example was learned in MNIST, permutedMNIST, and CIFAR-10 respectively. Rescaled view where the number of examples has been capped between 0 and 1500 for visualization purposes. Unforgettable examples are generally learned early during training, thus may be considered as "easy" and may have a low loss during most of the training.

the uncertainty of a classifier. Given an image, an image classifier outputs a probability for each class that the image may belong to. Directly from Shannon, entropy is defined over the probability space of the classifier's logits as

$$H(X) = -\sum_{x \in \mathcal{X}} p_\theta(x) \log\, p_\theta(x). \tag{2.14}$$

Another metric that can be used to inform us about a classifier's uncertainty is the classification margin. The classification margin m, slightly modified from (TSC19b) is the difference between the logit of the correct class and the largest logit among the other classes. Formally we have,

$$m = p(x)_k - \text{argmax}_{k' \neq k} p(x)_{k'}, \tag{2.15}$$

where k is the index corresponding to the correct class. Margin is only sensitive to the delta between the two highest predicted probabilities and is not sensitive to the rest of the distribution like entropy. These two metrics will be used throughout this dissertation.

Conclusions The forgettability score provides a means of comprehending the significant points that facilitate generalization and illuminates the training dynamic requisite for successful supervised learning. However, computing the forgettability score is computationally infeasible, and thus cannot be directly employed to expedite training. In this dissertation, we will compare forgettability scores to other methods of importance ordering, conduct a direct comparison of forgettability scores computed by first- and second-order optimizers, and calculate the entropy and classification margin over the logits to apprise us of the classifier's uncertainty.

CHAPTER 3

Preconditioned Stochastic Gradient Descent

We present a novel approach to accelerate stochastic gradient descent (SGD) by utilizing curvature information obtained from Hessian-vector products or finite differences of parameters and gradients, similar to the BFGS algorithm. Our approach involves two preconditioners: a matrix-free preconditioner and a low-rank approximation preconditioner. We update both preconditioners online using a criterion that is robust to stochastic gradient noise and does not require line search or damping. To preserve the corresponding symmetry or invariance, our preconditioners are constrained to certain connected Lie groups. The Lie group's equi-variance property simplifies the preconditioner fitting process, while its invariance property eliminates the need for damping, which is commonly required in second-order optimizers. As a result, the learning rate for parameter updating and the step size for preconditioner fitting are naturally normalized, and their default values work well in most scenarios. Our proposed approach offers a promising direction for improving the convergence of SGD with low computational overhead. We demonstrate that Preconditioned SGD (PSGD) outperforms SoTA on Vision, NLP, and RL tasks across multiple modern deep-learning architectures.

3.1 Introduction

Optimizing machine learning models with millions of free parameters presents a significant challenge. While conventional convex optimization algorithms such as Broyden-Fletcher-Goldfarb-Shanno (BFGS), its limited-memory version, L-BFGS, conjugate gradient (CG), and nonlinear versions like Hessian-free (HF) optimization (MS12) have succeeded in small-scale, convex mathematical optimization problems, they are rarely used for large-scale, stochastic optimization problems that arise in machine learning (ML). One of the main reasons is their reliance on the line search step. In many ML models, such as variational and reinforcement learning models, cost functions are defined as expectations and can only be evaluated through Monte Carlo (MC) sampling averages. This can result in large variances, making optimizers that rely on line search to ensure convergence problematic. Several recent extensions of these methods to deep learning, such as K-BFGS and KFAC, have foregone the line search step in favor of damping (GRB20; MG15a). However, this adds complexity by introducing extra hyperparameters.

Empirical results indicate that plain SGD is a highly efficient optimizer for most ML problems. However, for problems with a large eigenvalue spread, SGD may converge slowly once the solution is located in a basin of attraction. Regret optimizers such as RMSProp and Adam (KB15) converge faster but have been shown to generalize worse on many problems (WRS17; ZFM20). Reducing the generalization gap between SGD and Adam remains an active topic of research (ZTD20). This work focuses on providing SGD with a good preconditioner to accelerate its convergence around the basin of attraction without undermining its generalization capacity. The curvature information for preconditioner fitting can be sampled from Hessian-vector products or finite differences of parameters and gradients, similar to the BFGS algorithm. However, constructing a preconditioner in a deterministic way, as in BFGS (BV04a; GRB20), may not be possible due to potential issues with line search and damping. Therefore, we adopt the more general and gradient-noise-robust preconditioner fitting criterion proposed in (Li15) and fit the preconditioner online with another "gradient descent" algorithm. The key is to avoid making the preconditioner fitting problem more difficult and computationally expensive than the original parameter-learning problem.

In this disseration, we propose using Lie groups as a tool for preconditioner fitting. The "gradient descent" on a Lie group is similar to common gradient descent in Euclidean space.

It involves applying a series of small transforms via multiplication with $I + \mu G$, where μ is a small scalar and G is the group generator (see A.1.4.1). The Lie group is a rich and convenient space to work in since moving a preconditioner around any point on the group behaves similarly to moving it around the identity element of the group, i.e., the identity matrix I. This is known as the Lie group equivariance property.

Recent curvature-aware optimization methods such as HessianFree, KFAC, AdaHessian, K-LBFGS, and Shampoo have shown good empirical results in Deep Learning (OIY22). Yet, they require damping, line search, or regret and are thus susceptible to pitfalls that do not affect PSGD.

In an empirical setting, PSGD simultaneously surpasses the corresponding SoTA optimizers across vision, natural language processing (NLP), and reinforcement learning (RL) tasks, and estabishes new SoTA for many networks and settings, *e.g.* ResNet, LSTMs (HS97) and GPT-2 (RWC19). We consider optimization problems involving MNIST (LC10), CIFAR-10 (Kri09), Penn Treebank (MSM93), the complete works of Shakespeare, the Open Web Text (OWT) dataset (GC19), HalfCheetah and RoboWalker (BCP16). PSGD outperforms SoTA methods with negligible overhead compared to SGD across a wide range of optimization problems. PSGD provides practitioners with a powerful, stable, and efficient optimization tool that can significantly enhance the performance of deep learning models in various domains.

3.2 Background

3.2.1 Notations

The objective is to minimize a loss function defined as an expectation, $f(\theta) = E_z[\ell(\theta, z)]$, where $\theta \in \mathbb{R}^n$ is the parameter vector to be optimized and z is a random vector that can be sampled to evaluate the loss $\ell(\theta, z)$. We assume that the considered problem is second-order differentiable. To simplify the notation, we use $\hat{f}(\theta)$ to denote a sampled noisy evaluation of $f(\theta)$. A step of SGD with learning rate μ and an optional positive definite preconditioner P is given by:

$$\theta_{i+1} = \theta_i - \mu P \, \partial \hat{f}(\theta)/\partial \theta \mid_{\theta=\theta_i} \tag{3.1}$$

where i is the iteration index, $\mu > 0$ is the learning rate, and P typically is a variable or adaptive preconditioner. Once the solution enters a basin of attraction centered at a local minimum θ^*, we can approximate the iteration step in (3.1) as:

$$\theta_{i+1} - \theta^* \approx (I - \mu P \hat{H})(\theta_i - \theta^*) \tag{3.2}$$

where $\hat{H} = \frac{\partial^2 \hat{f}(\theta)}{\partial \theta^T \partial \theta} \mid_{\theta=\theta^*}$ is the sampled Hessian at the local minimum. Conceivably, the eigenvalue spread of $P\hat{H}$ largely determines the speed of convergence of the quasi-linear system in (3.2). Nearly quadratic convergence is possible if we can find a good approximation for H^{-1}. However, $\hat{H}$ is a noisy Hessian and is not necessarily positive definite, even if the exact one at θ^*, i.e., H, is.

3.2.2 The Preconditioner Fitting Criterion

We adopt the preconditioner fitting criterion proposed in (Li15). Let δg be the perturbation of gradient associated with parameter perturbation $\delta\theta$. Then, the fitting criterion is:

$$c(P) = E_{\delta\theta}[\delta g^T P \delta g + \delta\theta^T P^{-1} \delta\theta] \tag{3.3}$$

With auto differentiation tools, we can replace the pair $(\delta\theta, \delta g)$ with $(v, \hat{H}v)$, where v is a random vector, and $\hat{H}v$ is the noisy Hessian-vector product, which can be evaluated as cheaply as gradients. Criterion (3.3) only has one positive definite solution, $P = (H^2 + E_v[\epsilon^2])^{-\frac{1}{2}}$, even for indefinite H, where $\epsilon = \hat{H} - H$ is a stochastic noise term. This preconditioner automatically dampens gradient noise. It is worth noting that criterion (3.3) gives the same preconditioner used in equilibrated SGD (ESGD) (DVB15) and AdaHessian (YGS21) when P is diagonal, i.e., $E[v \odot v] \oslash E[(\hat{H}v) \odot (\hat{H}v)]$, where $\odot$ and $\oslash$ denote element-wise product and division, respectively.

3.2.3 Preconditioners on Lie Groups

It is natural to fit the preconditioner on a Lie group for several reasons. First, by rewriting equation (3.1) as $P^{-\frac{1}{2}}\theta_{i+1} = P^{-\frac{1}{2}}\theta_i - \mu\partial\hat{f}(\theta)/\partial(P^{-\frac{1}{2}}\theta) \mid_{\theta=\theta_i}$, it is clear that a preconditioned SGD is equivalent to SGD in a new set of coordinates defined by $\vartheta = P^{-\frac{1}{2}}\theta$. This coordinate change consists of rotations and scalings, i.e., operations on the orthogonal group $O(n)$ and the group of nonsingular diagonal matrices. We can represent this coordinate transform with matrix Q^{-1} and, accordingly, $P = Q^TQ$. Thus, we pursue a variable Q on the Lie group to fit this coordinate transform.

Second, PSGD can also be viewed as SGD with transformed features when the parameters to be learned are a list of affine transform matrices (Li19). Specifically, the most commonly used feature transformations (e.g., whitening, normalization, and scaling) can be represented as matrices on the affine groups. For example, the popular batch normalization (IS15), layer normalization (BKH16), and group normalization (WH18) can be represented as a sparse Lie group matrix where only the diagonal and last column can have nonzero values (Li19) (See A.1.1). The decorrelated batch normalization (HYL18) is related to the whitening preconditioner in (Li19). Thus, the Lie group arises as a natural object to work with.

Lie groups have two properties that are particularly suitable for our task. Like any group, a specific Lie group preserves certain symmetries or invariances. For example, with $Q \in GL^+(n,\mathbb{R})$, the general linear group with positive determinant, ϑ, and θ will always have the same orientation. This eliminates the need for damping, or similar remedies, to avoid degenerate solutions, since $P = Q^TQ$ is guaranteed to be invertible. The equivariance property of Lie groups further facilitates the preconditioner fitting. The same group generator, i.e., the one at the identity matrix, can be used to move a preconditioner on any point of the Lie group.

In fact, the preconditioner P estimated by PSGD converges to the inverse of the Hessian regardless of the definiteness of Hessian. From this, one can show that the parameters converge following the established results in the open literature. For more details and proofs see A.1.1.

The preconditioners proposed in (Li19) can only be applied to a list of affine transform matrix parameters. Although many machine learning models exclusively consist of affine transforms and nonlinear activation functions, this is not always the case. Additionally, it can be impractical to reparameterize many existing modules, such as a convolutional layer, into their equivalent affine transform forms. Hence, in this dissertation, we propose two types of novel black box preconditioners.

3.3 Curvature-Informed SGD via General Purpose Lie-Group Preconditioners

3.3.1 Simple Matrix Free Preconditioners

Lie groups are a type of abstract mathematical object that consists of transformations in vector space, specifically mappings that take vectors in $\mathbb{R}^n$ and map them to other vectors in the same space. Mathematically we have, $T : \mathbb{R}^n \mapsto \mathbb{R}^n$. The following theorem gives one systematic way to construct such sparse Lie group preconditioners.

Theorem 3.3.1. *Let $K = \{\sigma_1, \ldots, \sigma_m\}$ be a subgroup of the permutation group S_n. Then, linear transform $T : \mathbb{R}^n \mapsto \mathbb{R}^n$, $\quad T(x|a_1, \ldots, a_m) = \sum_{i=1}^{m} a_i \odot \sigma_i(x)$, forms a subgroup of $GL(n, \mathbb{R})$ parameterized with $\{a_1, \ldots, a_m\}$ if $T(\cdot|a_1, \ldots, a_m)$ is bijective, where both a_i and x are in $\mathbb{R}^n$.*

See proof of Theorem 3.3.1 in Appendix A.1.2.

Example 1: the group of invertible diagonal matrices. We must have $K = \{e\}$ if $|K| = 1$, where e is the identity element of S_n, i.e., $e(x) = x$. Then, T simply has a diagonal matrix representation, i.e., $T(x|a_1) = \operatorname{diag}(a_1)x$. Criterion (3.3) gives the preconditioner in ESGD (DVB15) and AdaHessian (YGS21) as a special case when P is on this group.

Example 2: The group of "X-shape matrices." Let $K = \{e, \sigma_f\}$, where σ_f denotes the flipping permutation. Then, we can show that

$$\begin{aligned} T(\cdot|a,b)T(\cdot|u,v) &= T(\cdot|a \odot u + b \odot \sigma_f(v), a \odot v + b \odot \sigma_f(u)) \\ T^{-1}(\cdot|a,b) &= T(\cdot|\sigma_f(a) \oslash c, -b \oslash c) \end{aligned}$$

where $c = a \odot \sigma_f(a) - b \odot \sigma_f(b)$. Clearly, such transforms form a Lie group if they are invertible, i.e., no element of c is zero. The matrix representation of this T only has nonzero diagonal and anti-diagonal elements, thus the name X-shape matrix.

Example 3: The butterfly matrix. For an even n, subgroup $K = \{e, s_{\frac{n}{2}}\}$ induces a Lie group whose representations are invertible butterfly matrices, where $s_{\frac{n}{2}}$ denotes circular shifting by $\frac{n}{2}$ positions. This group of matrices are the building blocks of the Kaleidoscope matrices (DSG20).

Example 4: The plain dense invertible matrix. The group $GL(n, \mathbb{R})$ can be recovered by letting $K = \{e, s_1, \ldots, s_{n-1}\}$, where s_i denotes circular shifting by i positions.

The group $GL(n, \mathbb{R})$ is too expensive for large-scale problems. The group of diagonal matrices also called the "Jacobi preconditioner" in its matrix form, is sparse but empirically shown to be less effective without the help of momentum for certain machine learning problems (DVB15). We are mostly interested in the cases with $2 \le |K| \le 4$. These Lie groups are sparse enough, yet simple enough to derive their inverse manually, and at the same time can significantly accelerate the convergence of SGD by shortcutting gradients separated far away in positions.

3.3.2 Low-Rank Approximation Preconditioner

Low-rank approximation (LRA) is a standard technique for processing large-scale matrices. Commonly adopted forms of positive definite low-rank approximation, such as $P = \rho I + UU^T$, cannot always be factorized as $P = Q^TQ$ for certain Lie groups, where $\rho > 0$ is a small positive number. Additionally, this form of approximation is not effective for reducing the eigenvalue spread. In many real-world problems, the Hessian has a few very large and very small eigenvalues, i.e., tails on both ends of the spectrum (SBL16; SEG17). However, all the eigenvalues of P in this form are lower bounded by ρ, meaning that it can only fit one tail of the spectrum when $\text{rank}(U) \ll n$.

For this reason, we propose a new low-rank approximation with form $Q = \rho(I + UV^T)$, where ρ is not necessarily small nor positive, and U and V has r columns with $r \ll n$. To justify this form of approximation, we need to establish two facts. First, preconditioner $P = Q^TQ$ with this form can fit both tails of the spectra of the Hessian, providing an accurate characterization of the curvature of a function, improving optimization algorithms, and permitting assessment of their robustness. Second, we can update this preconditioner on Lie groups.

Theorem 3.3.2. *Preconditioner $P = Q^TQ$ with $Q = \rho(I + UV^T)$ can have positive eigenvalues arbitrarily larger than ρ^2 and arbitrarily smaller than ρ^2 with proper U and V.*

Theorem 3.3.3. *If $\rho \neq 0$ and $(I + V^TU)^{-1}$ or $(I + U^TV)^{-1}$ exists, $A_V(\rho, U) = \rho(I + UV^T)$ defines a subgroup of $GL(n, \mathbb{R})$ parameterized with ρ and U. Similarly, $A_U(\rho, V) = \rho(I + UV^T)$ defines another subgroup of $GL(n, \mathbb{R})$ parameterized with ρ and V.*

See proofs of Theorem 3.3.2 & 3.3.3 in Appendix A.1.3 & A.1.3.

Table 3.1: Test accuracy of LeNet5 on MNIST over 10 runs.

SGD	Adam	KFAC	PSGD_{A}	$\text{PSGD}_{\text{XMat}}$	PSGD_{LRA}
99.04	99.12	99.16	99.26	99.22	99.22

Table 3.2: Stage & cosine learning rate and residual-free ResNet18-RF on CIFAR10.

	ResNet18		ResNet18-RF	
lr	*cos*	*stage*	*cos*	*stage*
SGD	95.51	95.07	94.97	94.35
PSGD	95.54	95.43	95.36	95.17

3.4 Practical Considerations

The above-proposed preconditioners can be fit online by minimizing criterion (3.3) using gradient descent on Lie groups. Unlike traditional gradient descent, moving an object on a Lie group is achieved by multiplying it with $I + \mu G$, where G is the group generator and μ is small enough such that $|\mu G| < 1$. This series of small movements trace a curve on the Lie group manifold, and G is always in the tangent space of the group as the Lie algebra is closed. See A.1.4 for mathematical considerations.

Note that optimizer damping is neither necessary nor generally feasible on any Lie group, although it is widely used in other second-order optimizers to avoid degenerate solutions. On one hand, by fitting Q on a connected Lie group, $P = Q^T Q$ cannot be singular. On the other hand, damping could be incompatible with certain forms of Lie groups, as we may not always be able to find another Q' on the same group such that $Q'^T Q' = Q^T Q + \lambda I$, where $\lambda > 0$. This eliminates the need for setting up a proper damping schedule. However, gradient clipping can be helpful in promoting stability. The quadratic approximation leading to the quasi-linear system (3.2) is only valid within a certain region around θ. Thus, $\|\delta\theta\| = \mu \|P \partial \hat{f}(\theta)/\partial\theta\|$ should be small enough such that $\theta + \delta\theta$ still locates in this trust region. We can adjust μ or clip $\|P \partial \hat{f}(\theta)/\partial\theta\|$ to ensure that $\|\delta\theta\|$ is small enough.

Lastly, the learning rate for parameter updating and step size for preconditioner fitting are naturally normalized to be in the range $(0, 1)$. We have found a step size of 0.01 to be an effective initial rate, and our method is robust to a wide range of learning rates and weight decays. (see Appendix B.1.2.1)

3.5 Empirical Results

In this work, we evaluate the performance of the PSGD algorithm on a diverse set of tasks, including vision, natural language processing (NLP), and reinforcement learning (RL). In the domain of computer vision, we evaluate the algorithm's performance on the MNIST (LC10) dataset and using a LeNet5 (see Table 3.1) and a convex large-scale logistic regression (see Appendix B.1.3). Aditionally we consider the CIFAR10 (Kri09) dataset (see Table 3.2) with ResNet18. Finally, we consider several variants of the CIFAR10 dataset, including those with noisy labels, blur-deficit, adversarial attacks, and class imbalances (see Table 3.3).

For NLP tasks, we study the use of LSTMs (HS97) and GPT-2 style transformers (RWC19), on various text corpora, including the Penn Treebank (MSM93), the complete works of Shakespeare, and the Open Web Text (OWT) dataset (GC19) (see Table 3.4 & 3.5). In the RL setting, we consider a Proximal Policy Optimization (PPO) (SWD17) applied to the HalfCheetah and RoboWalker environmets using OpenAI's gym environments (BCP16) (see Fig. 3.1).

To provide insight into the fundamental differences between SGD and PSGD, we perform uncertainty and forgettability analysis (TSC18). Finally, we examine the pathological delayed XOR problem, first introduced in (HS97), in order to further our understanding of the strengths and limitations of different optimization methods.

3.5.1 CIFAR-10 and Friends on ResNet18

We examine the effectiveness of the PSGD optimizer in classifying CIFAR10-derived datasets using a ResNet18 and find it outperforms the SoTA optimizer SGD. Once the optimizers are tuned on ResNet18 with the CIFAR10 dataset, we do not change them for the rest of the experiments.

CIFAR10 ResNet18 We evaluate the performance of PSGD and SGD on the CIFAR10 image classification task using the ResNet18 model, following the implementation of the AdaBelief algorithm (ZTD20). We adopt the cosine learning rate scheduler and observe that SGD achieves the SOTA test accuracy of 95.5%. Our results (see Table 3.2) show that PSGD slightly outperforms SGD when the shortcut connections are removed or a less-tuned learning rate scheduler is used. Both optimizers are tuned with hyperparameters such as learning rate and weight decay and use a momentum of 0.9. We update the preconditioner of PSGD every ten iterations, resulting in a negligible overhead compared to SGD.

Removing Skip Connections To increase curvature in the relatively flat modern ResNet architecture, we remove the residual skip connections that gave ResNets their namesake. A recent study (ZBM22) demonstrated the use of Tailored Activation Transformation (TAT) in conjunction with K-FAC (MG15a)(another second-order optimizer) to close the gap between residual networks with and without residual skip connections. However, in our experiment, we do not utilize TAT and instead compare the performance of SOTA optimizers on both residual-free and standard ResNet18 models. The results, summarized in Table 3.2, indicate that PSGD outperforms SGD by 0.61% and 0.20% on residual-free and standard ResNet18, respectively. Our findings are consistent with the results from (ZBM22), where a difference of 0.6% was observed between the optimization of residual-free using TAT and standard ResNet18.

Modified CIFAR10 Next, we evaluate the performance of SGD and PSGD on three variations of the CIFAR10 image classification task using ResNet18: class imbalance, noisy training labels, and adversarial generalization. These variations provide a rigorous test for the generalization ability of optimizers. We maintain the hyperparameters from our previous experiments on the ResNet18 CIFAR10 dataset for a fair comparison between the optimizers.

Class Imbalanced CIFAR10 We evaluate the optimization performance on a class-imbalanced version of the CIFAR10 dataset, where 50% of the classes are randomly reduced by an order of magnitude. We compare SGD and PSGD on optimizing ResNet18 and report the results in Table 3.3. Our results show that PSGD outperforms the SOTA by 1.03% on this dataset.

Table 3.3: Accuracy of ResNet18 on diverse CIFAR10 derived tasks using PSGD vs SGD.

CIFAR10	Standard	No Shortcut	Class Imb	Noisy Final	Noisy Best	Adversarial	Blur Deficit
PSGD + M	$95.49_{0.08}$	$95.27_{0.09}$	$87.16_{0.85}$	$78.63_{0.11}$	$78.63_{0.11}$	$85.17_{0.04}$	$89.51_{0.12}$
SGD +M	$95.47_{0.14}$	$94.66_{0.16}$	$86.32_{0.84}$	$26.9_{33.75}$	$73.975_{5.65}$	$83.82_{0.18}$	$83.51_{0.15}$

CIFAR10 with Noisy Labels We randomly select 60% of the CIFAR10 training labels, resulting in each class having approximately 46% correct labels and the 54% consisting of 6% drawn from the other nine classes. We used SGD and PSGD to train a ResNet18 on this task for 100 epochs with 5 different seeds. Both optimizers achieved a training accuracy of around 44%. However, their test accuracies showed a significant difference. Among the 5 runs of SGD, only one *lucky initilization* achieved a test accuracy of 77%, while the other initializations had a test accuracy of 10% with an average predicted probability of 0.999. This is not a standard case of over-fitting nor a case of catastrophic forgetting, since the training accuracy does not change. Instead, our experiments show there exists a tipping point in the test accuracy at around epoch 35, where within a single epoch the test accuracy drops from 71% to 10% while the training accuracy shows no indication of this change. Furthermore, the test accuracy does not increase for the rest of the 65 epochs. PSGD achieved an average accuracy of 78.63% after 200 epochs, outperforming SGD by 51.73% and exhibiting no optimization instabilities (See Table 3.3).

Adversarial Attacked CIFAR10 Finally, we trained a ResNet18 on 10k unmodified CIFAR10 images and evaluated it on a test set consisting of 40k samples perturbed using Neural Tangent Generalization Attacks (YW21). As shown in Table 3.3, PSGD outperformed SGD by 1.35%.

3.5.2 Language Modeling

We conduct a performance comparison between second-order optimization methods and first-order methods for training recurrent neural networks (RNNs) and transformer models. RNNs have a recurrent structure that exhibits strong curvature properties, making them particularly suitable for second-order optimization methods. Such methods can efficiently

Table 3.4: Test perplexity (lower is better) on Penn Treebank for one-, two- and three-layered LSTMs. All results except PSGD in the table are reported by AdaBelief and Adan.

LSTM	PSGD	Adan	AdaBelief	SGD	AdaBound	Adam	AdamW	Padam	RAdam	Yogi
1 layer	**83.5**	83.6	84.2	85.0	84.3	85.9	84.7	84.2	86.5	86.5
2 layers	**64.9**	65.2	66.3	67.4	67.5	67.3	72.8	67.2	72.3	71.3
3 layers	**59.7**	59.8	61.2	63.7	63.6	64.3	69.9	63.2	70.0	67.5

leverage this structure to converge quickly to good solutions. Consequently, RNNs usually perform better when trained with second-order optimization methods (Mar16), particularly when the objective function has extreme local curvature properties.

In contrast, transformers employ sparse self-attention mechanisms that allow for efficient computation of gradients. Thus, they are typically trained using first-order methods. Our aim is to provide a comprehensive comparison of these optimization methods for both RNN and transformer models.

LSTM based models We benchmark PSGD by predicting the Penn TreeBank Dataset using LSTMs using (ZTD20)'s framework. Our results (see Table 3.4) indicate that PSGD consistently outperforms (lower is better) other first-order optimization algorithms, including the SoTA AdamW, on 1, 2, and 3 layer LSTMs.

nanoGPT In a recent study, (Kar22) proposed a framework for reproducing GPT-2 results using the OpenWebText dataset. Here, we expand upon this by investigating the training of GPT-2 models of different sizes on both the OpenWebText and Shakespeare character datasets. Our primary objective is to benchmark the performance of PSGD against other SoTA optimization algorithms.

As shown in Table 3.5, our results indicate that PSGD consistently outperforms other optimization algorithms across various model sizes. Notably, a moderate gap in perplexity is observed for the smaller GPT-2 model trained for 5k iterations on the Shakespeare-char dataset, with a significantly larger gap observed on the OpenWebText dataset, which was trained for 600k iterations using AdamW and 100k iterations using PSGD. We found that decreasing the *precond lr* to 0.001 greatly improved the performance of transformer models.

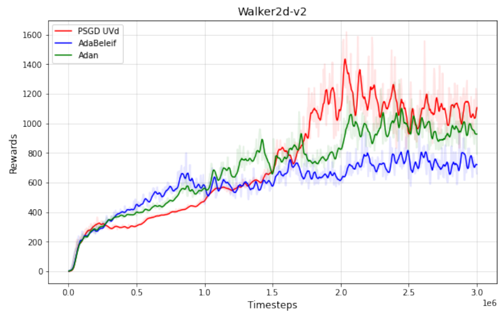
Table 3.5: Comparing different test loss of optimizers over GPT-2 style transformers on the Shakespeare-char dataset. PSGD outperforms other optimizers over various model sizes.

nanoGPT	PSGD	SGD	AdamW	AdanW	AdaBelief	AdanBelief
0.82M	**4.52**	4.68	4.68	5.52	5.94	6.66
1.61M	**4.47**	4.75	5.03	5.054	5.06	6.47
6.37M	**4.53**	5.31	19.53	4.92	21.04	5.34
50M	**250.7**		591.8			

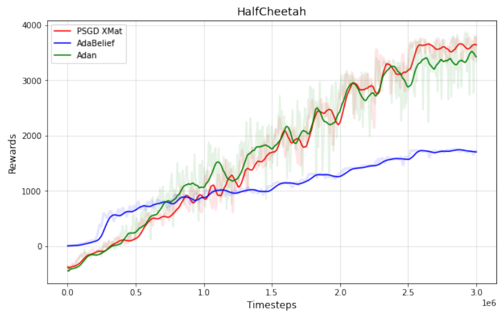
Lowering the *precond lr* smoothens the curvature in the sparse embedding layer of GPT-2 over time and enables the optimizer to consider a larger window of curvature information. This 'curvature memory' improved performance and prevented divergence resulting from the otherwise sparse embedding space.

3.5.3 Reinforcement Learning

Here we consider two standard Proximal Policy Optimization (PPO) problems in Reinforcement Learning (RL): Walker2d and HalfCheetah. We compare the performance of the two SOTA optimizers AdaBelief and Adan to PSGD. We optimize the actor and critic independently. We find that PSGD can find a higher reward for both Walker2d & HalfCheetah shown in Figure 3.1.



(a) **Walker2d**



(b) **HalfCheetah**

Figure 3.1: PSGD outperforms SOTA optimizers on PPO RL on Walker2d & HalfCheetah.

3.5.4 Towards Understanding Second Order Optimizers

With the performance distinctions between PSGD and SGD now apparent, we aim to conduct a series of simple experiments to provide further insight into the reasons behind the stark contrast in the nature of the two optimizers.

Uncertainty Analysis We train a standard ResNet18 on CIFAR10 with PSGD and SGD, and after 200 epochs we check the entropy over the softmax-logits and classification margin (TSC18) of both NNs. We see in Table 3.6 that PSGD has higher entropy and a lower margin of classification compared to first-order optimizers. This very low minimum entropy of first-order optimizers may be interpreted as a form of overfitting. Intuitively, some data points (low entropy/unforgettable) have information that is easily memorized by NNs, giving up network capacity learning points that do not improve generalization. Conversely, we observe that PSGD never becomes overly certain about any data point, with the minimum entropy being six orders of magnitude larger than that of SGD. Similar trends can be observed in the mean and maximum entropy values. In terms of margin, PSGD on average has a 95% certainty level for classifications, while SGD has a staggering 99.9%. From these statistics, we believe first-order optimizers can push NNs into a dangerous regime of overconfidence which given clean labels can reduce their generalization capacity with the potential of catastrophic forgetting. In the scenario where a network is given noisy labels (or imaginably poisoned points), training with first-order methods may lead to memorization of the training set with no generalization capacity as seen in Table 3.3 column Noisy Final.

PSGD's higher entropy and lower margins are indications of a larger exploration of the parameter space, more diverse solutions, and a lack of memorization. This suggests that PSGD is able to better balance the trade-off between overfitting and underfitting, without memorizing points.

Forgettability Statistics and Learning We revisit (TSC18)'s forgettability experiments, which found that one can prune 30% of the CIFAR-10 train samples without loss of generalization. The study found that a point's utility for generalization increases as it is learned and then subsequently forgotten during training, regardless of the optimizer or architecture used. Essentially, forgettability ordering shows which data points a NN uses to define high-dimensional boundaries akin to support vectors. We investigate whether the performance difference between PSGD and SGD's generalization performance can be attributed to this forgettability ordering. Furthermore, (TSC18) revealed a strong correlation between forgettable points and high margin, which we explore by examining the expected forgettability ordering of the two optimizers and whether pruning points based on their orderings affects performance.

We train the ResNet18 three times pruning CIFAR10 by 25k, 35k, and 45k points based on each optimizer's forgettability score. Table 3.7 shows that points PSGD focuses on based on forgettability ordering outperform that of SGD. When we limit the dataset to only the 5k most forgettable datapoints, we see PSGD is able to outperform SGD by nearly 14%.

SGD and PSGD exhibit fundamentally different importance orderings, which is evident from the significant generalization gap observed when training on pruned datasets using these orderings at various degrees. We find that PSGD and SGD have a statistically significant correlation between their forgettability orderings, but the strength of the correlation is weak (Spearman coefficient of 0.02 and a p-value of $p = 1\text{x}10^{-12}$). This indicates that the nature of training observed through forgettability is different for PSGD compared to first-order optimizers.

Furthermore, we see a strong correlation coefficient of 0.75 and 0.60 between a low forgetability score and low entropy score with a p-value of $p = 0$ for PSGD and SGD respectively (see Fig B.5). Hence, PSGD does a better job of shaping the NN to focus on the highly forgettable points that are important for generalization while not becoming overconfident on the easy unforgettable points giving up too much capacity, unnecessarily pushing the parameters away from initialization reducing generalization ability (CG19) (see B.3). Note while (TSC18) shows that forgettable points are more important for generalization for supervised

Table 3.6: Uncertainty Statistics: PSGD's higher uncertainty leads to better generalization and less over-fitting.

NTK	**Entropy**			**Margin**		
	Min	Mean	Max	Min	Mean	Max
PSGD	0.139	0.260	1.545	0.144	0.956	0.994
SGD	7x10^{-7}	0.01	0.8975	0.3925	0.999	1

Table 3.7: Forgetting statistics for CIFAR10 on ResNet18. PSGD finds better forgettability statistics outperforming SGD.

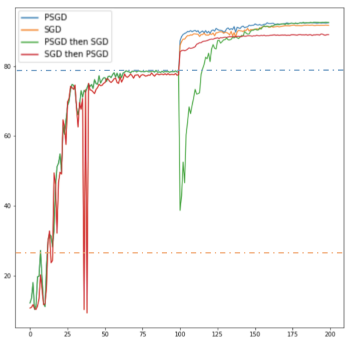
Forgetting	**50k**	**25k**	**15k**	**5k**
PSGD	96.65	95.83	94.95	56.46
SGD	96.21	95.56	93.7	42.48

learning, very recently (PKN23) showed low entropy points are the more important points for learning in the unsupervised, semi-supervised, and generative model setting. See Table B.3 showing generalization gap training low and high entropy points and how it affects the distance from initialization.

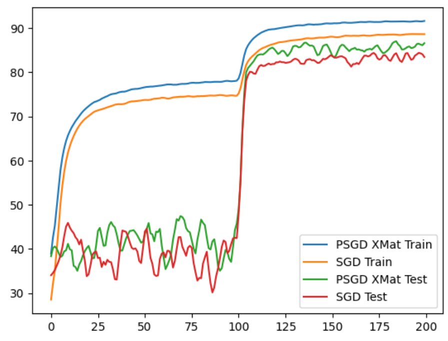
Stubborn Solutions of First Order Optimizers From the previous examples, we learned that first-order optimizers tend to heavily overfit certain data points, reducing entropy. In the case of noisy labels, this often results in pure memorization where the network achieves the training accuracy of PSGD on the train set, but only 10% accuracy on the test dataset with an average confidence of 99.9%. To better understand the stubbornness of SGD solutions, we consider the *lucky initialization,* which resulted in a test accuracy of 77% under noisy label conditions. Here, we examine a transfer learning problem, where a ResNet18 was trained on noisy data for 100 epochs and then on clean labels for another 100 epochs. This simulates a real-world distributional shift where noisy labels may be refined throughout training, leading to a distributional shift. We compare neural networks trained with each optimizer, as well as those that changed optimizers after 100 epochs of training.

As seen in Fig 3.2a, PSGD finds a flexible solution when given noisy labels. This is seen since when we correct the labels both PSGD and SGD can reach an accuracy of 92.42%. In contrast, the solution found by SGD seems to be stubborn since when we correct the noisy labels, PSGD then SGD reaches an accuracy of 91.7% and SGD then PSGD has an accuracy of 88%. This shows the importance of curvature information in the early periods of training.

Intuitively, first-order methods cause NNs to dedicate parameters to memorizing some data points, unnecessarily reducing entropy. When memorization occurs given incorrect labels, it may be difficult to reshape the NN when the labels are corrected, leading to the loss in generalization accuracy seen in Fig 3.2a. In contrast, as PSGD finds a less certain or stubborn solution in the first stage of training, either optimizer can then reshape the NN to their liking in the second stage.



(a) **Noisy Label CIFAR-10**



(b) **Nero-Plasticity**

Figure 3.2: a) Solutions found by SGD are harder to optimize compared to PSGD. Note for SGD we used a lucky initialization (see 3.5.1). The blue and yellow dotted line are the average accuracy of PSGD and SGD after 100 epochs respectivly. b) Removing the blur-deficit at epoch 100, PSGD is be more neuro-plastic compared to SGD, achieving better train and test performance.

PSGD Preserves Neuro-Plasticity Recently, (ARS17) studied the phenomenon of neuro-plasticity in NNs. This presents a different lens to our stubborn solution hypothesis. They found that NNs exhibit a critical learning period, during which if a learning deficit is not removed, the NN becomes unable to learn effectively. To simulate this deficit, CIFAR10 images are blurred for the first half of training, after which the deficit is removed. Following (ARS17), we used an exponential learning rate decay. To investigate the impact of optimization on neuro-plasticity we compare SGD and PSGD. We find that PSGD retains neuro-plasticity outperforming SGD by 6 and 3% on test and train sets seen in Fig 3.2b and Table 3.3.

We believe the noisy-label and neuro-plasticity results have strong applicability to transfer learning and coreset selection (PDM22a), particularly in scenarios where the training distribution changes during training. Recently, (OIY22) demonstrated that the affine Kronecker variant of PSGD outperforms other first and second-order optimizers when fine-tuning a ResNet18 and ViT (DBK20) on CIFAR10 that were pre-trained on ImageNet (DDS09).

Understanding long-term dependence via Delayed XOR Problem We consider the delayed XOR problem proposed in the LSTM paper (HS97). The task is to predict the XOR relation of a and b scattered randomly and far away in a long sequence. This problem is challenging for many architectures and optimizers because it cannot be 'partially solved' since memorizing either a or b alone does not help to predict XOR(a, b). We consider solving this problem with standard RNNs and LSTMs optimized using SGD, AdaBelief, Adan, AdaHessian, Hessian Free, Shampoo, KFAC, and PSGD at different sequence lengths. The success rates for each optimizer are shown in Table 3.8. In our experiments, we find that PSGD LRA is the only method that can solve the XOR problem past the length of 32 with an RNN. Furthermore, LSTMs show no benefit to RNNs without using curvature information for the XOR problem. Finally, PSGD optimizes an RNN to solve the XOR problem better than any of the other methods using an LSTM. These result hint, that while increasing model parameters and architecture complexity may boost performance, the choice of optimizer may be more important for certain problems. See B.1.5 for rank analysis.

XOR	**PSGD LRA**		**AdaHessian**		**KFAC**		**HessianFree**		**Shampoo**		**SGD**		**AdaBelief**		**Adan**	
Length	RNN	LSTM	RNN	LSTM	RNN	LSTM	RNN	LSTM	RNN	LSTM	RNN	LSTM	RNN	LSTM	RNN	LSTM
32	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
55	1	1	0	1	0	0	0	0.6	0	0.8	0	0	0	0	0	0
64	1	1	0	0.8	0	0	0	0	0	0.6	0	0	0	0	0	0
112	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 3.8: Delayed XOR problem at length 32,55,64, and 112 averaged over 10 runs. An RNN optimized with PSGD outperforms other optimizers with an RNN or LSTM. This hints that the choice of optimizer may be more important than network architecture.

3.6 Conclusion

This chapter has presented a comprehensive study of black-box Lie group preconditioners for the optimization of deep learning problems. We have provided theoretical guarantees for the convergence of Lie group preconditioned optimization methods, and empirical results demonstrating PSGD outperforming SoTA optimizers in generalization, robustness, and stability across various tasks in Vision, NLP, and RL. Furthermore, our analysis of forgettability and entropy shows that PSGD effectively focuses on forgettable points, leading to improved generalization. These findings provide valuable insights for further developing efficient and effective optimization techniques for deep learning models.

CHAPTER 4

Adaptive Curvature-Informed Coreset Selection

Training machine learning models on massive datasets incurs substantial computational costs. To alleviate such costs, there has been a sustained effort to develop data-efficient training methods that can carefully select subsets of the training examples that generalize on par with the full training data. However, existing methods are limited in providing theoretical guarantees for the quality of the models trained on the extracted subsets and may perform poorly in practice. We propose AdaCore, a method that leverages the geometry of the data to extract subsets of the training examples for efficient machine learning. The key idea behind our method is to dynamically approximate the curvature of the loss function via an exponentially-averaged estimate of the Hessian to select weighted subsets (coresets) that provide a close approximation of the full gradient preconditioned with the Hessian. We prove rigorous guarantees for the convergence of various first and second-order methods applied to the subsets chosen by AdaCore. Our extensive experiments show that AdaCore extracts coresets with higher quality compared to baselines and speeds up the training of convex and non-convex machine learning models, such as logistic regression and neural networks, by over 2.9x over the full data and 4.5x over random subsets.

4.1 Motivation for Coreset Selection

Large datasets have been crucial for the success of modern machine-learning models. Learning from massive datasets, however, incurs substantial computational costs and becomes very challenging (AD19; SGM19; SDS19). Crucially, not all data points are equally important for learning (BMB19; KF18; TSC18). While several examples can be excluded from training without harming the accuracy of the final model (BMB19; TSC18), other points need to be trained on many times to be learned (BMB19). To improve scalability of machine learning, it is essential to theoretically understand and quantify the value of different data points on training and optimization. This allows identifying examples that contribute the most to learning and safely excluding those that are redundant or non-informative.

To find essential data points, recent empirical studies used heuristics such as the fully trained or a smaller proxy model's uncertainty (entropy of predicted class probabilities) (CYM20), or forgetting events (TSC18) to identify examples that frequently transition from being classified correctly to incorrectly. Others employ either the gradient norm (ALS15; KF18) or the loss (LH15; SQA15) to sample important points that reduce the variance of stochastic optimization methods. Such methods, however, do not provide any theoretical guarantee for the quality of the trained model on the extracted examples.

Quantifying the importance of different data points without training a model to convergence is very challenging. First, the value of each example cannot be measured without updating the model parameters and measuring the loss or accuracy. Second, as the effect of different data points changes throughout training, their value cannot be precisely measured before training converges. Third, to eliminate redundancies, one needs to look at the importance of individual data points as well as the higher-order interactions between data points. Finally, one needs to provide theoretical guarantees for the performance and convergence of the model trained on the extracted data points.

Here, we focus on finding data points that contribute the most to learning and automatically excluding redundancies while training a model. A practical and effective approach is to carefully select a small subset of training examples that closely approximate the full gradient, i.e., the sum of the gradients over all the training data points. This idea has been recently employed to find a subset of data points that guarantee convergence of first-order methods to near-optimal solutions for training convex models (MBL20). However, modern machine learning models are high dimensional and non-convex in nature. In such scenarios, subsets selected based on gradient information only capture gradients along the sharp dimensions and lack diversity within groups of examples with similar training dynamics. Hence, they represent large groups of examples with a few data points with substantial weights. This introduces a large error in the gradient estimation and results in first-order coresets that perform poorly.

We propose *ADAptive second-order COREsets* (AdaCore) that incorporates the geometry of the data to iteratively select weighted subsets (coresets) of training examples that capture the gradient of the loss preconditioned with the Hessian, by maximizing a submodular function. Such subsets capture the curvature of the loss landscape along different dimensions and provide convergence guarantees for first and second-order methods. As a naive use of Hessian at every iteration is prohibitively expensive for overparameterized models, AdaCore relies on Hessian-free methods to extract coresets that capture the full gradient preconditioned by the Hessian diagonal. Furthermore, AdaCore exponentially averages first and second-order information in order to smooth the noise in the local gradient and curvature information.

We first provide a theoretical analysis of our method and prove its convergence for convex and non-convex functions. For a β-smooth and α-strongly convex loss function and a subset S selected by AdaCore that estimates the full preconditioned gradient by an error of at most ϵ, we prove that Newton's method and AdaHessian applied to S with constant stepsize $\eta = \alpha/\beta$ converges to a $\beta\epsilon/\alpha$ neighborhood of the optimal solution, in exponential rate. For non-convex overparameterized functions such as deep networks, we prove that for a β-smooth and μ-PL* loss function satisfying $\|\nabla\mathcal{L}(w)\|^2/2 \geq \mu\mathcal{L}(w)$, (stochastic) gradient descent applied to subsets found by AdaCore has similar training dynamics to that of training on full data, and converges at a exponential rate. In both cases, AdaCore leads to a speedup by training on smaller subsets.

Next, we empirically study the examples selected by AdaCore during training. We show that as training continues, AdaCore selects more uncertain or forgettable samples. Hence, AdaCore effectively determines the value of every learning example, i.e., when and how many times a sample needs to be trained on, and automatically excludes redundant and non-informative instances. Importantly, incorporating curvature in selecting coresets allows AdaCore to quantify the value of training examples more accurately, and find fewer but more diverse samples than existing methods.

We demonstrate the effectiveness of various first and second-order methods, namely SGD with momentum, Newton's method and AdaHessian, applied to AdaCore for training models with a convex loss function (logistic regression) as well as models with a non-convex loss function, namely ResNet-20, ResNet-18, and ResNet-50, on MNIST, CIFAR10, (Imbalanced) CIFAR100, and BDD100k (Den12)(Kri09)(YCW20). Our experiments show that AdaCore can effectively extract crucial samples for machine learning, resulting in higher accuracy while achieving over 2.9x speedup over the full data and 4.5x over random subsets, for training models with convex and non-convex loss functions.

4.2 Related Work

Data-efficient methods have recently gained a lot of interest. However, existing methods often require training the original (BMB19; GZ19; TSC18) or a proxy model (CYM20) to convergence, and use features or predictions of the trained model to find subsets of examples that contribute the most to learning. While these results empirically confirm the existence of notable semantic redundancies in large datasets (BMB19), such methods cannot identify the crucial subsets before fully training the original or the proxy model on the entire dataset. Most importantly, such methods do not provide any theoretical guarantees for the model's performance trained on the extracted subsets.

There have been recent efforts to take advantage of the difference in importance among various samples to reduce the variance and improve the convergence rate of stochastic optimization methods. Those that are applicable to overparameterized models employ either the gradient norm (ALS15; KF18) or the loss (LH15; SQA15) to compute each sample's importance. However, these methods do not provide rigorous convergence guarantees and cannot provide a notable speedup. A recent study proposed a method, Craig, to find subsets of samples that closely approximate the full gradient, i.e., the sum of the gradients over all the training samples (MBL20). Craig finds the subsets by maximizing a submodular function and provides convergence guarantees to a neighborhood of the optimal solution for strongly-convex models. GradMatch (KSM21) proposes a variation to address the same objective using orthogonal matching pursuit (OMP) (KSM21), and Glister Killamsetty(KSR20) aims at finding subsets that closely approximate the gradient of a held-out validation set. However, Glister requires a validation set, and GradMatch uses OMP which may return subsets as little as 0.1% of the intended size. Such subsets are then augmented with random samples. In contrast, our method successfully finds subsets of higher quality by preconditioning the gradient by the Hessian information.

4.3 AdaCore: Background and Problem Setting

Training machine learning models often reduces to minimizing an empirical risk function. Given a not-necessarily convex loss $\mathcal{L}$, one aims to find model parameter vector w_* over the parameter space $\mathcal{W}$ that minimizes the loss $\mathcal{L}$ over the training data V:

$$w_* \in \arg\min_{w\in\mathcal{W}}\mathcal{L}(w), \tag{4.1}$$
$$\mathcal{L}(w) := \sum_{i\in V} l_i(w), \quad l_i(w) = l(f(x_i, w), y_i),$$

where $V = \{1, \dots, n\}$ is an index set of the training data, $w \in \mathbb{R}^d$ is the parameters of the model f being trained, and l_i is the loss function associated with training example $i \in V$ with feature vector $x_i \in \mathbb{R}^d$ and label y_i. We denote the gradient of the loss w.r.t. model parameters by $\mathbf{g} = \nabla\mathcal{L}(w) = \frac{1}{|V|}\sum_{i\in V}\frac{\partial l_i}{\partial w}$, and the corresponding second derivative (i.e., Hessian) by $\mathbf{H} = \nabla^2\mathcal{L}(w) = \frac{1}{|V|}\sum_{i\in V}\frac{\partial^2 l_i}{\partial w_j \partial w_k}$. First-order gradient methods are popular for solving Problem (4.1). Such methods start from an initial point w_0, and every iteration t steps in the negative direction of the gradient. The most popular first-order method, Stochastic Gradient Descent (SGD) (RM51), is often used with momentum accelerating it in directions whose gradients point in the same directions and dampens oscillations in dimensions whose gradients change directions (Qia99):

$$w_{t+1} = w_t - \eta_t \mathbf{v}_t, \qquad \mathbf{v}_t = \mathbf{g}_t, \tag{4.2}$$

Here, η_t is the learning rate, and $\mathbf{g}_t$ is the gradient of a batch at the t-th iteration.

4.4 AdaCore: Adaptive Second order Coresets

The key idea behind our proposed method is to leverage the geometry of the data, precisely the curvature of the loss landscape, to select subsets of the training examples that enable fast convergence. Here, we first discuss why coresets that only capture the full gradient perform poorly in various scenarios. Then, we show how to incorporate curvature information in subset selection for training convex and non-convex models with provable convergence guarantees— ameliorating problems of first-order coresets.

4.4.1 When First-order Coresets Fail

First-order coreset methods iteratively select weighted subsets of training data that closely approximate the full gradient at particular values of w_t, e.g. beginning of every epoch (KSM21; KSR20; MBL20):

$$S_t^* = \underset{S\subseteq V, \gamma_{t,j}\geq 0\ \forall j}{\arg\min} |S| \quad \text{s.t.} \quad \|\mathbf{g}_t - \sum_{j\in S}\gamma_{t,j}\mathbf{g}_{t,j}\| \leq \epsilon, \tag{4.3}$$

where $\mathbf{g}_{t,j}$ and $\gamma_{t,j} > 0$ are the gradient and the weight of element j in the coreset S. Such subsets often perform poorly for high-dimensional and non-convex functions, due to the following reasons: (1) the scale of gradient $\mathbf{g} \in \mathbb{R}^d$ is often different along different dimensions. Hence, the selected subsets estimate the full gradient closely only along dimensions with a larger gradient scale. This can introduce a significant error in the optimization trajectory for both convex and non-convex loss functions; (2) the loss functions associated with different data points l_i may have similar gradients but very different curvature properties at a particular w_t. Thus, for a small $\delta > 0$, the gradients $\nabla l_i(w_t + \delta)$ at $w_t + \delta$ may be totally different than the gradients $\nabla l_i(w_t)$ at w_t. Consequently, subsets that capture the gradient well at at a particular point during training may not provide a close approximation of the full gradient after a few gradient updates, e.g., mini-batches. This often results in inferior performance, particularly when selecting larger subsets for non-convex loss functions; (3) subsets that only capture the gradient, select one representative example with a large weight from data points with similar gradients at w_t. Such subsets lack diversity and cannot distinguish different subgroups of the data. Importantly, the large weights introduce a substantial error in estimating the full gradient and result in poor performance, as we show in Fig. B.8 in the Appendix.

4.4.2 Adaptive Second-order Coresets

To address the above issues, our main idea is to select subsets of training examples that capture the full gradient preconditioned with the curvature of the loss landscape. In doing so, we normalize the gradient by multiplying it by the Hessian inverse, $\mathbf{H}^{-1}\mathbf{g}$, before selecting the subsets. This allows selecting subsets that (1) can capture the full gradient in all dimensions equally well; (2) contain a more diverse set of data points with similar gradients, but different curvature properties, and (3) allow adaptive first and second-order methods trained on the coresets to obtain similar training dynamics to that of training on the full data.

Formally, our goal in AdaCore is to adaptively find the smallest subset $S \subseteq V$ and corresponding per-element weights $\gamma_j > 0$ that approximates the full gradient preconditioned with the Hessian matrix, with an error of at most $\epsilon > 0$ at every iteration t, I.e.:

$$S_t^* = \underset{S \subseteq V, \gamma_{t,j} \geq 0 \ \forall j}{\arg\min} |S|, \quad \text{s.t.} \tag{4.4}$$
$$\|\mathbf{H}_t^{-1}\mathbf{g}_t - \sum_{j \in S} \gamma_{t,j} \mathbf{H}_{t,j}^{-1} \mathbf{g}_{t,j}\| \leq \epsilon,$$

where $\mathbf{H}_t^{-1}\mathbf{g}_t$ and $\sum_{j \in S} \gamma_{t,j} \mathbf{H}_{t,j}^{-1} \mathbf{g}_{t,j}$ are preconditioned gradients of the full data and the subset S.

4.4.3 Scaling up to Over-parameterized Models

Directly solving the optimization problem (4.4) requires explicit calculation and storage of the Hessian matrix and its inverse. This is infeasible for large models such as neural networks. In the following, we first address the issue of calculating the inverse Hessian at every iteration. Then, we discuss how to efficiently find a near-optimal subset to estimate the full preconditioned gradient by solving Eq. (4.4).

Approximating the Gradients For neural networks, the derivative of the loss $\mathcal{L}$ w.r.t. the input to the last layer (KF18; MBL20) or the penultimate layer (KSM21) can capture the variation of gradient norm well. We extend these results (Appendix A.2.4.2) to show that the normed difference preconditioned gradient difference between data points can be approximately efficiently bounded by:

$$\|\mathbf{H}_i^{-1}\mathbf{g}_i - \mathbf{H}_j^{-1}\mathbf{g}_j\| \leq c_1 \|\Sigma_L'(z_i^{(L)})(\mathbf{H}_i^{-1}\mathbf{g}_i)^{(L)} - \Sigma_L'(z_j^{(L)})(\mathbf{H}_j^{-1}\mathbf{g}_j)^{(L)}\| + c_2, \tag{4.5}$$

where $\Sigma_L'(z_i^{(L)})(\mathbf{H}_i^{-1}\mathbf{g}_i)^{(L)}$ is gradient preconditioned by the inverse of the Hessian of the loss w.r.t. the input to the last layer for data point i, and c_1, c_2 are constants. Since the upper bound depends on the weight parameters, we need to update our subset S using AdaCore during the training.

Calculating the last layer gradient often requires only a forward pass, which is as expensive as calculating the loss, and does not require any extra storage. For example, having a softmax as the last layer, the gradients of the loss w.r.t. the i^{th} input to the softmax is $p_i - y_i$, where p_i is the i^{th} output the softmax and y is the one-hot encoded label with the same dimensionality as the number of classes. Using this low-dimensional approximation $\hat{\mathbf{g}}_i$ for the gradient $\mathbf{g}_i$ we can efficiently calculate the preconditioned gradient for every data point. For non-convex functions, the local gradient information can be very noisy. To smooth out the local gradient information and get a better approximation of the global gradient, we apply an exponential moving average with a parameter $0 < \beta_1$ to the low-dimensional gradient approximations:

$$\overline{\mathbf{g}}_t = \frac{(1-\beta_1)\sum_{i=1}^{t} \beta_2^{t-i} \hat{\mathbf{g}}_{\mathbf{i}}}{1-\beta_2^t}. \tag{4.6}$$

Approximating the Hessian Preconditioner Since it is infeasible to calculate, store, and invert the full Hessian matrix every iteration, we use an inexact Newton method, where an approximate Hessian operator is used instead of the full Hessian. To efficiently calculate the Hessian diagonal, we first use the Hessian-Free method (YXR18) to compute the multiplication between Hessian $\mathbf{H}_t$ and a random vector z with Rademacher distribution. To do so, we backpropagate on the low-dimensional gradient estimates multiplied by z to get $\mathbf{H}_t z = \partial \hat{\mathbf{g}}_t^T z / \partial w_t$. Now, we can use Hutchinson's method of obtaining a stochastic estimate of the diagonal of the Hessian matrix as follows:

$$\text{diag}(\mathbf{H}_t) = \mathbb{E}[z \odot (\mathbf{H}_t z)], \tag{4.7}$$

without having to form the Hessian matrix explicitly (BKS07). The diagonal approximation has the same convergence rate as using the Hessian for strongly convex, and strictly smooth functions (Proof in Appendix A.2.1). Nevertheless, our method can be applied to general machine learning problems, such as deep networks and regularized classical methods (e.g., SVM, LASSO), which are strongly convex. To smooth out the noisy local curvature and get a better approximation of the global Hessian information, we apply an exponential moving average with parameter $0 < \beta_2 < 1$ to the Hessian diagonal estimate in Eq. (4.7):

$$\overline{\mathbf{H}}_t = \sqrt{\frac{(1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i}\text{diag}(\mathbf{H}_i)\text{diag}(\mathbf{H}_i)}{1-\beta_2^t}}. \tag{4.8}$$

Using exponentially averaged gradient and the Hessian approximations in Eq. (4.6), and (4.8), the preconditioned gradients in Eq. (4.4) can be approximated as follows:

$$S_t^* = \underset{S\subseteq V, \gamma_{t,j}\geq 0\ \forall j}{\arg\min} |S|, \quad \text{s.t.} \tag{4.9}$$
$$\|\overline{\mathbf{H}}_t^{-1}\overline{\mathbf{g}}_t - \sum_{j\in S}\gamma_{t,j}\overline{\mathbf{H}}_{t,j}^{-1}\overline{\mathbf{g}}_{t,j}\| \leq \epsilon.$$

Next, we discuss how to efficiently find near-optimal weighted subsets that closely approximate the full preconditioned gradient by solving Eq. (4.4).

4.4.4 Extracting Second-order Coresets

The subset selection problem (4.4) is NP-hard (Nat95). However, it can be considered as a special case of the sparse vector approximation problem that has been studied in the literature, including convex optimization formulations—e.g. basis pursuit (CDS01), sparse projections (PEC12; KBC13), LASSO (Tib96), and compressed sensing (Don06). These methods, however, are expensive to solve and often require tuning regularization coefficients and thresholding to ensure cardinality constraints. More recently, the connection between sparse modeling and *submodular*[1] optimization have been demonstrated (EKD18; MBL20). The advantage of submodular optimization is that a fast and simple greedy algorithm often provides a near-optimal solution. Next, we briefly discuss how submodularity can be used

[1]A set function $F : 2^V \to \mathbb{R}^+$ is submodular if $F(S\cup\{e\})-F(S) \geq F(T\cup\{e\})-F(T)$, for any $S \subseteq T \subseteq V$ and $e \in V \setminus T$.

to find a near-optimal solution for Eq. (4.4). We build on the recent result of (MBL20) that showed that the error of estimating an expectation by a weighted sum of a subset of elements is upper-bounded by a submodular facility location function. In particular, via the above result, we get:

$$\begin{aligned}\min_{S\subseteq V}\|\overline{\mathbf{H}}_t^{-1}\overline{\mathbf{g}}_t-\sum_{j\in S}\gamma_{t,j}\overline{\mathbf{H}}_{t,j.}^{-1}\overline{\mathbf{g}}_{t,j}\ \| \\ \leq\sum_{i\in V}\min_{j\in S}\|\overline{\mathbf{H}}_{t,i.}^{-1}\overline{\mathbf{g}}_{t,i}-\overline{\mathbf{H}}_{t,j.}^{-1}\overline{\mathbf{g}}_{t,j}\|.\end{aligned}\tag{4.10}$$

Setting the upper bound in the right-hand side of Eq. (4.10) to be less than ϵ results in the smallest weighted subset S^* that approximates full preconditioned gradient by an error of at most ϵ, at iteration t. Formally, we wish to solve the following optimization problem:

$$\begin{aligned}S^*\in&\arg\min\nolimits_{S\subseteq V}|S|,\quad \text{s.t.}\\ L(S)=&\sum_{i\in V}\min_{j\in S}\|\overline{\mathbf{H}}_{t,i.}^{-1}\overline{\mathbf{g}}_{t,i}-\overline{\mathbf{H}}_{t,j.}^{-1}\overline{\mathbf{g}}_{t,j}\|\leq\epsilon,\end{aligned}\tag{4.11}$$

By introducing a phantom example e, we can turn the minimization problem (4.11) into the following submodular cover problem, with a facility location objective $F(S)$:

$$\begin{aligned}S^*\in\arg\min_{S\subseteq V}|S|,\quad \text{s.t.}\\ F(S)=C_1-L(S\cup\{e\})\geq C_1-\epsilon,\end{aligned}\tag{4.12}$$

where $C_1=L(\{e\})$ is a constant upper-bounding the value of $L(S)$. The subset S^* obtained by solving the maximization problem (4.12) is the medoid of the preconditioned gradients, and the weights γ_j is the number of elements that are closest to the medoid $j\in S^*$, i.e. $\gamma_j=\sum_{i\in V}\mathbb{I}[j=\min_{s\in S}\|\overline{\mathbf{H}}_{t,i}^{-1}\overline{\mathbf{g}}_{t,i}-\overline{\mathbf{H}}_{t,s}^{-1}\overline{\mathbf{g}}_{t,s}\|]$. For the above submodular cover problem, the classical greedy algorithm provides a logarithmic approximation guarantee $|S|\leq\big(1+\ln(\max_e F(e|\emptyset))\big)|S^*|$ (Wol82). The greedy algorithm starts with the empty set

$S_0 = \emptyset$, and at each iteration t, it chooses an element $e \in V$ that maximizes the marginal utility $F(e|S_t) = F(S_t \cup \{e\}) - F(S_t)$. Formally, $S_t = S_{t-1} \cup \{\arg\max_{e \in V} F(e|S_{t-1})\}$. The computational complexity of the greedy algorithm is $\mathcal{O}(nk)$. However, its complexity can be reduced to $\mathcal{O}(|V|)$ using stochastic methods (MBK15), and can be further improved using lazy evaluation (Min78) and distributed implementations (MKS13). The pseudocode can be found in Alg. 3 in Appendix A.2.3.

One coreset for convex functions For convex functions, normed gradient differences between data points can be efficiently upper-bounded by the normed difference between feature vectors (AYS16; HLL15; MBL20). We apply a similar idea to upper-bound the normed difference between preconditioned gradients. This allows us to find one subset before the training. See proof in Appendix A.2.4.1.

4.4.5 Convergence Analysis

Here, we analyze the convergence rate of first and second-order methods applied to the weighted subsets S found by AdaCore. By minimizing Eq. (4.12) at every iteration t, AdaCore finds subsets that approximate the full preconditioned gradient by an error of at most ϵ, i.e. $\|\mathbf{H}_t^{-1}\mathbf{g}_t - \sum_{j \in S} \gamma_{t,j} \mathbf{H}_{t,j}^{-1} \mathbf{g}_{t,j}\| \leq \epsilon$. This allows us to effectively analyze the reduction in the value of the loss function $\mathcal{L}$ at every iteration t. Below, we discuss the convergence of a first and second-order gradient methodapplied to subsets extracted by AdaCore.

Convergence for Newton's Methods and AdaHessian We first provide the convergence analysis for the case where the function $\mathcal{L}$ in Problem (4.1) is strongly convex, i.e. there exists a constant $\alpha > 0$ such that $\forall w, w' \in \mathbb{R}^d$ we have $\mathcal{L}(w) \geq \mathcal{L}(w') + \langle \nabla \mathcal{L}(w'), w - w' \rangle + \frac{\alpha}{2}\|w' - w\|^2$, and each component function has a Lipschitz gradient, i.e. $\forall w \in \mathcal{W}$ we have $\|\nabla \mathcal{L}(w) - \nabla \mathcal{L}(w')\| \leq \beta \|w - w'\|$. We get the following results by applying Newton's method and AdaHessian to the weighted subsets S extracted by AdaCore.

Theorem 4.4.1. *Assume that $\mathcal{L}$ is α-strongly convex and β-smooth. Let S be a weighted subset obtained by* AdaCore *that estimates the preconditioned gradient by an error of at most ϵ at every iteration t, i.e., $\|\mathbf{H}_t^{-1}\boldsymbol{g}_t - \sum_{j\in S}\gamma_{t,j}\mathbf{H}_{t,j}^{-1}\boldsymbol{g}_{t,j}\| \leq \epsilon$. Then with learning rate α/β, Newton's method with update rule of Eq.* (2.8) *applied to the subsets has the following convergence behavior:*

$$\mathcal{L}(w_{t+1}) - \mathcal{L}(w_t) \leq -\frac{\alpha^3}{2\beta^4}(\|\mathbf{g}_t\| - \beta\epsilon)^2. \tag{4.13}$$

In particular, the algorithm converges to a $\beta\epsilon/\alpha$-neighborhood of the optimal solution w_*.

Corollary 4.4.2. *For an α-strongly convex and β-smooth loss $\mathcal{L}$, AdaHessian with Hessian power k, applied to subsets found by* AdaCore *converges to a $\beta\epsilon/\alpha$-neighborhood of the optimal solution w_*, and satisfies:*

$$\mathcal{L}(w_{t+1}) - \mathcal{L}(w_t) \leq -\frac{\alpha^{k+2}}{2\beta^{k+3}}(\|\mathbf{g}_t\| - \beta\epsilon)^2. \tag{4.14}$$

The proofs can be found in Appendix A.2.1.

Convergence for (S)GD in Over-parameterized Case Next, we discuss the convergence behavior of gradient descent applied to the subsets found by AdaCore. In particular, we build upon the recent results of (LZB20) that guarantees convergence for first-order methods on a broad class of general over-parameterized non-linear systems, including neural networks for which the tangent kernel, defined as $\mathbf{J}^T\mathbf{J}$ are not close to constant but satisfy the Polyak-Lojasiewicz (PL) condition. Where $\mathbf{J} = \partial f/\partial w$ is the Jacobian of the function f with respect to the parameters w. A loss function $\mathcal{L}$ is μ-PL* on a set $\mathcal{W}$, if $\frac{1}{2}\|\nabla\mathcal{L}(w)\|^2 \geq \mu\mathcal{L}(w), \forall w \in \mathcal{W}$.

Theorem 4.4.3. *Assume that the loss function $\mathcal{L}(w)$ is β-smooth, and μ-PL* on a set $\mathcal{W}$, and S is a weighted subset obtained by* AdaCore *that estimates the preconditioned gradient* *by an error of at most ϵ, i.e., $\|\mathbf{H}_t^{-1}\boldsymbol{g}_t - \sum_{j\in S}\gamma_{t,j}\mathbf{H}_{t,j}^{-1}\boldsymbol{g}_{t,j}\| \leq \epsilon$. Then with learning rate η, gradient descent with update rule of Eq.* (4.2) *applied to the subsets have the following convergence behavior at iteration t:*

$$\mathcal{L}(w_t) \leq (1 - \frac{\eta\mu\alpha^2}{\beta^2})^t\mathcal{L}(w_0) - \frac{\eta\alpha^2}{2\beta^2}(\beta^2\epsilon^2 - 2\beta\epsilon\nabla_{\max}), \tag{4.15}$$

where α is the minimum eigenvalue of all Hessian matrices during training, and $\nabla_{\max}$ is an upper bound on the norm of the gradients.

Theorem 4.4.4. *Under the same assumptions as in Theorem 4.4.3, for mini-batch SGD with mini-batch size $m \in \mathbb{N}$, the mini-batch SGD with update rule Eq.* (4.2)*, with learning rate $\eta = \frac{m}{\beta(m-1)}$, applied to the subsets have the following convergence behavior:*

$$\mathbb{E}[\mathcal{L}(w_t)] \leq (1 - \frac{\eta\mu\alpha^2}{2\beta})^t\mathbb{E}[\mathcal{L}(w_0)] - \frac{\alpha^2\eta}{2\beta}(\beta\epsilon^2 - 2\epsilon\nabla_{\max}) \tag{4.16}$$

where α is the minimum eigenvalue of all Hessian matrices during training, and $\nabla_{\max}$ is an upper bound on the norm of the gradients, and the expectation is taken w.r.t. the randomness in the choice of mini-batch.

The proofs can be found in Appendix A.2.2.

We show an exponential convergence for GD (Theorem 4.4.3) and SGD (Theorem 4.4.4) under the μ-PL* condition, as well as for second order methods (Theorems 4.4.1, 4.4.2), under α-strongly convex and β-smooth assumptions on the loss.

4.5 Experiments

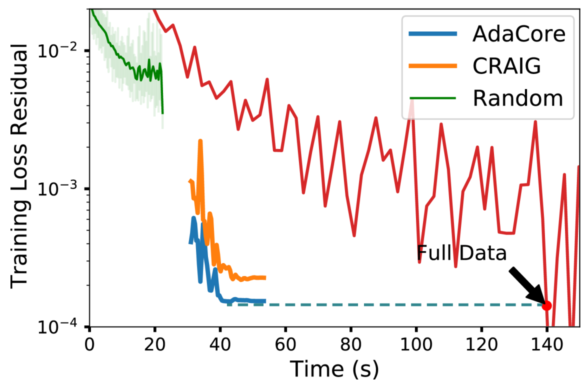
In this section, we evaluate the effectiveness of AdaCore, by answering the following questions: (1) how does the performance of various first and second-order methods compare when applied to subsets found by AdaCore vs. the full data and baselines; (2) how effective is AdaCore for extracting crucial subsets for training convex and non-convex over-parameterized models with different optimizers; and (3) how does AdaCore perform in eliminating redundancies and enhancing the diversity of the selected elements.

Baselines In the convex setting, we compare the performance of AdaCore with Craig (MBL20) that extract subsets that approximate the full gradient, as well as Random subsets. For non-convex experiments, we additionally compare AdaCore with GradMatch and Glister (KSM21; KSR20). For AdaCore and Craig, we use the gradient w.r.t to the input to the last layer, and for Glister and GradMatch we use the gradient w.r.t the penultimate layer, as specified by the methods. In all cases, we select subsets separately from each class proportional to the class sizes, and train on the union of the subsets. We report average test accuracy across 3 trials in all experiments.

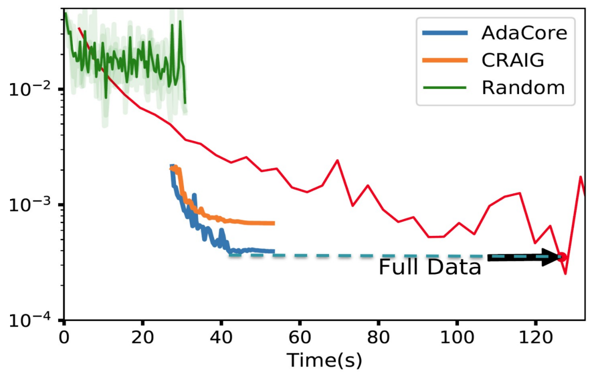
4.5.1 Convex Experiments

In our convex experiments, we apply AdaCore to select a coreset to classify the Ijcnn1 dataset using L2-regularized logistic regression: $f_i(x) = ln(1 - \exp(-w^T x_y y_i)) + 0.5\mu w^T w$. Ijcnn1 includes 49,990 training and 91,701 test data points of 22 dimensions, from 2 classes with a 9-to-1 class imbalance ratio. In the convex setting, we only need to calculate the curvature once to find one AdaCore subset for the entire training. Hence, we utilize the complete Hessian information, computed analytically, as discussed in Appendix A.2.5. We apply an exponential decay learning schedule $\alpha_k = \alpha_0 b^k$ with learning rate parameters α_0 and b. For each model and method (including the random baseline) we tuned the parameters via a search and reported the best results.

AdaCore achieves smaller loss residual with a speedup Figure 4.1 compares the loss residual for SGD and Newton's method applied to coresets of size 10% extracted by AdaCore (blue), Craig (orange), and random (green) with that of full dataset (red). We



(a) Ijcnn1 SGD



(b) Ijcnn1 Newton

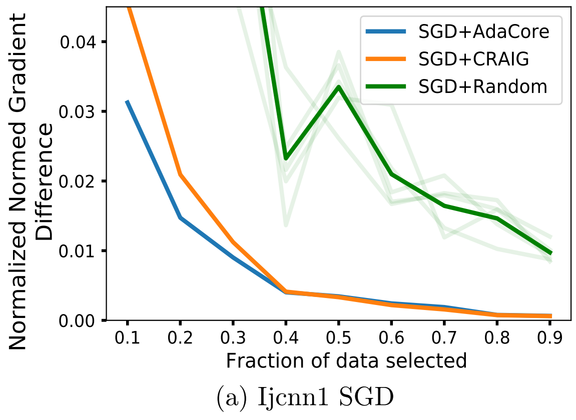
Figure 4.1: Loss residual of SGD and Newton's method for training Logistic Regression on Ijcnn1. Comparing AdaCore (blue), Craig (orange), and random subsets (green) of size 10% vs. entire data (red dot). AdaCore achieves 2.5x speedup for training with SGD and Newton's method.

see that AdaCore effectively minimizes the training loss, achieving a better loss residual than Craig and random sampling. In particular, AdaCore matches the loss achieved on the full dataset with more than a 2.5x speedup for SGD and Newton's methods. We note that training on random 10% subsets of the data cannot effectively minimize the training loss. We show the superior performance of training with SGD on subsets of size 10% to 90% found with AdaCore vs Craig in Appendix Fig. B.7.

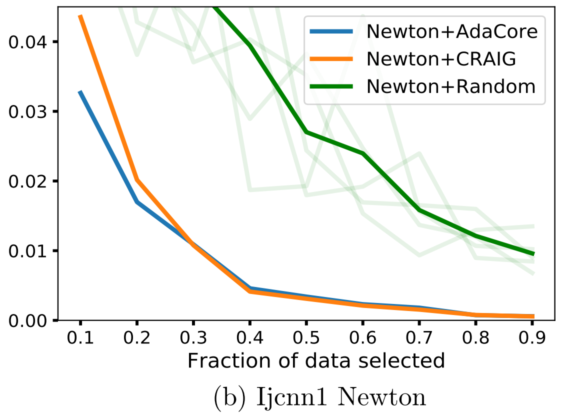
AdaCore better estimates the full gradient Fig. 4.2 shows the normalized gradient difference between the gradient of the full data vs. the weighted gradient of subsets of different sizes obtained by AdaCore vs Craig and Random, at the end of training by each method. We see that by considering curvature information, AdaCore obtains a better gradient estimate than Craig and Random subsets.

4.5.2 Non-Convex Experiments

Datasets We use CIFAR10 (60k points from 10 classes) , class imbalanced version of CIFAR10 (32.5k points from 10 classes) and CIFAR100 (32.5k points from 100 classes)



(a) Ijcnn1 SGD



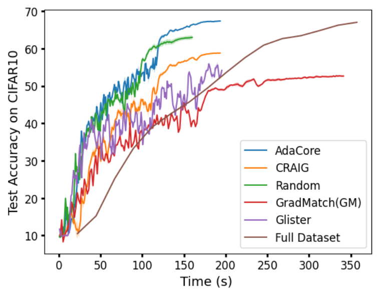
(b) Ijcnn1 Newton

Figure 4.2: Normalized gradient difference between subsets of various sizes found by AdaCore (blue), AdaCore (orange), Random (green) vs full data, when training Logistic Regression with SGD and Newton on Ijcnn1. AdaCore has a smaller gradient error at the end of training.

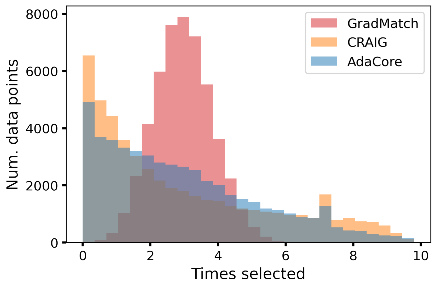
(Kri09), BDD100k (100k points from 7 classes) (YCW20). The results on MNIST (70k points from 10 classes) (Den12) can be found in Appendix B.2.6. Images are normalized to [0,1] by division with 255.

Models and Optimizers We train ResNet-20 and ResNet-18 (HZR16), with convolution, average pooling, and dense layers with softmax outputs and weight decay of 10^{-4}. We use a batch size of 256 in all experiments (except Table 4.3, Fig. 4.4a), and train using SGD with momentum of 0.9 (default), or AdaHessian. For training, we use a standard learning rate scheduler for ResNet starting with 0.1 and exponentially decaying by factor 0.1 at epochs 100 and 150. We used linear learning rate warm-up for the first 20 epochs to prevent weights from diverging when training with subsets. All experiments were run on a 2.4GHz CPU and RTX 2080 Ti GPU.

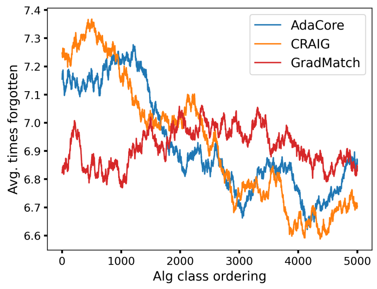
Calculating the Curvature To calculate the Hessian diagonal using Eq. (4.7), we use a batch size of $b_H = 64$ to calculate the expected Hessian diagonal over the training data. We observed that a smaller batch size provides a higher quality Hessian compared to larger batch sizes, as shown in Table 4.1.



(a) Test Accuracy



(b) Distribution of selected points



(c) Forgetting vs class ranking

Figure 4.3: Training ResNet-18 on subsets of size S=1% selected every R=1 epoch, with AdaCore, Craig, Glister, GradMatch and Random for 200 epochs vs. full for 15 epochs. (a) AdaCore outperforms baselines by providing 2x speedup over full, and more than 4.5x speedup over Random. (b) Histograms of the number of times a point is selected by AdaCore, Craig, and GradMatch. AdaCore selects a more diverse set of examples compared to Craig, and GradMatch augments several randomly selected examples. (c)Forgetting scores for examples of a class sorted by AdaCore at the end of training. AdaCore priorities less forgettable examples compared to Craig.

Baseline Comparison and Ablation Study Table 4.1 shows the accuracy of training ResNet-20, using SGD with the momentum of 0.9 and AdaHessian, for 200 epochs on S=1% subsets of CIFAR-10 chosen every R=1 epoch by different methods. For SGD+momentum, AdaCore outperforms Craig by 12%, Random by 10%, GradMatch by 6%, and Glister by 16.8%. Note that in total, AdaCore selects 74% of the dataset during the entire training process, whereas Random visits 87%. Thus, AdaCore effectively selects subsets contributing the most to generalization. We see that the accuracy gap between the baselines and AdaCore shrinks when applying more powerful optimizers such as AdaHessian. Table 4.1 also shows the effect of exponential averaging of gradients and Hessian diagonal, and larger batch sizes for calculating the Hessian diagonal b_H. We see that exponential averaging helpsAdaCore achieve better performance, and smaller b_H provides better results.

Fig. 4.3a compares the performance of ResNet-18 on 1% subsets selected from CIFAR-10 with different methods. We compare the performance of training on AdaCore, Craig, GradMatch, Glister, and Random subsets for 200 epochs, with training on full data for

	AdaHessian	**SGD+Momentum**
Random	59.1%± 2.8(87%)	45.9%± 2.5(87%)
CRAIG	59.5%± 2.8(74%)	43.6%± 1.6(75%)
GRADMATCH	57.5%± 1.3(74%)	49.4%± 1.6(74%)
GLISTER	37.5%± 1.3(74%)	38.6% ± 1.6(74%)
ADACORE (no avg)	58.4%± 0.2(73%)	51.5%± 1.1(74%)
ADACORE (avg g)	59.8%± 0.5(73%)	53.2% ± 1.1(74%)
ADACORE (avg H)	60.2%± 0.5(73%)	54.4%± 1.1(74%)
ADACORE	**60.2%**± 0.5(**73%**)	**55.4%**± 1.1(**74%**)
ADACORE b_h=512	57.2% ± 0.5(73%)	52.4% ± 1.1(74%)

Table 4.1: Training ResNet20 using AdaHessian and SGD+momentum on coresets of size 1% selected by different methods from CIFAR10. Percent of full data selected during the entire training is shown (in parentheses). Using b_H=64, ADACORE achieves up to 16.8% higher accuracy while selecting a smaller fraction of data points. Exponential averaging of gradient and Hessian, and a smaller b_H helps.

15 epochs. This is the number of iterations required for training on the full data to achieve comparable performance to that of ADACORE subsets. We see that training on ADACORE coresets achieves a better accuracy 2.5x faster than training on the full dataset, and more than 4.5x faster than the next best subset selection algorithm for this setting (*c.f.* Fig. B.9b in Appendix for complete results).

Frequency and size of subsets selection Table 4.2, 4.4 shows the performance of different methods for selecting subsets of size S% of the data every R epochs, from CIFAR-10 and imbalanced CIFAR-10. Table 4.2 shows that selecting subsets of size 1% every $R = 5, 10, 20$ epochs with ADACORE achieves superior performance compared to the baselines. Table

Table 4.2: Test accuracy and percent of full data selected (in parentheses), when selecting S=1% coresets every R epochs from Imbalanced CIFAR-10 to train ResNet18.

	S=1%, R=20	S=1%, R=10	S=1%, R=5
AdaCore	**57.3%(5%)**	**57.12(9.5%)**	**60.2%(14.5%)**
CRAIG	48.6%(8%)	55(16%)	53.05%(27.5%)
Random	54.7%(8%)	54.6(18%)	54.6%(33.2%)
GRADM	29.9%(8.2%)	29.1%(14.7%)	32.75%(23.2%)
GLISTER	21.1%(8.6%)	17.2%(16%)	14.4%(22.2%)

4.4 shows that AdaCore can successfully select larger subsets of size $S = 10\%, 30\%$ and outperform Craig (Std is reported in Appendix, Table B.4).

AdaCore speeds up training Fig 4.4 compares the speedup of various methods during training ResNet18 on 10% subsets selected every $R = 20$ epoch from BDD 100k and CIFAR-100. All the methods are trained to achieve a test accuracy between 72% and 74% on BDD 100k and between 57% and 50% on CIFAR-100. On BDD 100k, AdaCore achieves 74% accuracy in 100 epochs and training on full data achieves similar performance in 45 epochs. For CIFAR-100, AdaCore achieves 59% accuracy in 200 epochs and training on full data achieves a similar performance in 40 epochs. Complete results on speedup and test accuracy of each method can be found in Appendix B.2.3, B.2.4. We see that AdaCore achieves 2.5x speedup over training on full data and 1.7x over that of training on random subsets on BDD 100k. For CIFAR-100, AdaCore achieves 4.2x speedup over training on random subsets and 2.9x over training on full data. Compared to the baselines, AdaCore can achieve the desired accuracy much faster.

Effect of batch size Table 4.3 compares the performance of training with different batch sizes on subsets found by various methods. We see that training with larger batch size on subsets selected by AdaCore can achieve superior accuracy. As AdaCore selects more diverse subsets with smaller weights, one can train with larger mini-batches on the subsets

Table 4.3: Training ResNet18 with S=1% subsets every R=1 epoch from CIFAR10 using batch size b= 512, 256, 128. AdaCore can leverage larger mini-bath size and obtain a larger accuracy gap to Craig and Random. For b=512, we have 1 mini-batch (GD). Std is reported in Appendix Table B.8.

	AdaC.	Craig	Rand	Gap/ Craig	Gap/ Rand
GD b=512	**58.32**%	56.32%	49.14%	1.69%	**8.91**%
SGD b=256	**68.23**%	58.3%	60.7%	**9.93**%	8.16%
SGD b=128	**66.89**%	58.17%	65.46%	**8.81**%	1.52%

Table 4.4: Test accuracy and percent of full data selected (in parentheses), when selecting $S\%$ coresets every R epochs from CIFAR-10 and Imbalanced CIFAR-10 to train ResNet18.

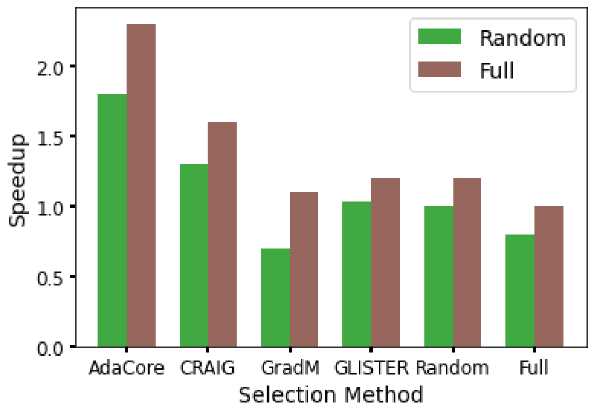
	ResNet20, CIFAR10 $S = 30\%$, $R = 20$	ResNet20, CIFAR10 $S = 10\%$, $R = 20$	ResNet18, CIFAR10-IMB $S = 30\%$, $R = 20$	ResNet18, CIFAR10-IMB $S = 10\%$, $R = 20$
AdaCore	**80.57%** ± 0.11 (**74.6%**)	**70.6%** ± 0.33 (**44.8%**)	**85.7%** ± 0.1 (**74%**)	**76%** ± 0.3 (**43.8%**)
Craig	65.8% ± 0.41 (90.9%)	58.5% ± 1.27 (60.75%)	79.3% ± 1.6 (84.5%)	71.6% ± 0.15 (56.4%)

without increasing the gradient estimate error. In contrast, Craig subsets have elements with larger weights and hence training with fewer larger mini-batches has a larger gradient error and does not improve the performance.

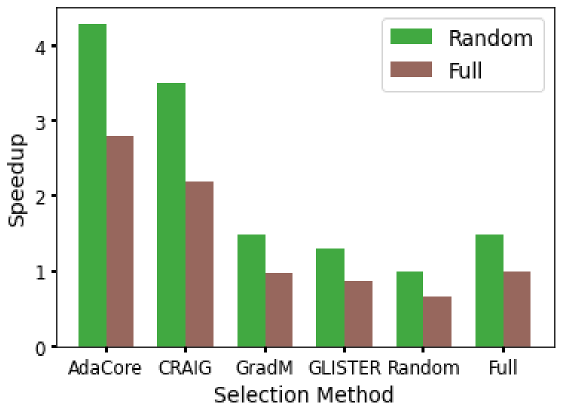
In summary, see that AdaCore consistently outperforms the baselines over various architectures, optimizers, subset sizes, selection frequency, and batch sizes.

AdaCore selects more diverse subsets Fig. 4.3b shows the number of times different methods selected particular elements during the entire training. We see that AdaCore successfully selects a more diverse set of examples compared to Craig. We note that Grad-Match may not be able to select subsets with the desired size, and instead augments the selected subset with randomly selected examples. Hence, it has a normal-shaped distribution. Fig. 4.3c shows the mean forgetting score for all examples within a class ranked by AdaCore at the end of the training, over a sliding window of size 100. We see that AdaCore prioritizes selecting less forgettable examples. This shows that indeed AdaCore is able to distinguish different groups of easier examples better, and hence can prevent catastrophic forgetting by including their representatives in the coresets.

AdaCore vs Forgettability and Uncertainty Fig. 4.5a, 4.5b show mean forgettability and uncertainty in sliding windows of size 100, 200 over examples sorted by AdaCore at the end of training. We see that AdaCore heavily biases its selections towards forgettable and uncertain points, as training proceeds. Interestingly, 4.5a reveals that AdaCore avoids the most forgettable samples in favor of slightly more memorable ones, suggesting that AdaCore can better distinguish easier groups of examples. Figure 4.5b shows similar bias



(a) BDD100k, ResNet50



(b) CIFAR100, ResNet18

Figure 4.4: Speedup of various methods over training on random subsets and full data, for training ResNet18 on CIFAR100 and ResNet50 on BDD100k with batch size=128.

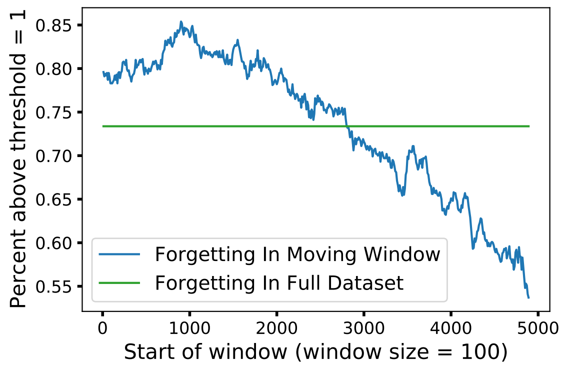
towards uncertain samples. Fig. 4.5c, 4.5d show the most and least selected images by AdaCore, respectively. We see the redundancies in the never selected images, whereas images frequented by AdaCore are quite diverse in color, angles, occluded subjects, and airplane models. This confirms the effectiveness of AdaCore in extracting the most crucial subsets for learning and eliminating redundancies.

4.6 Conclusion

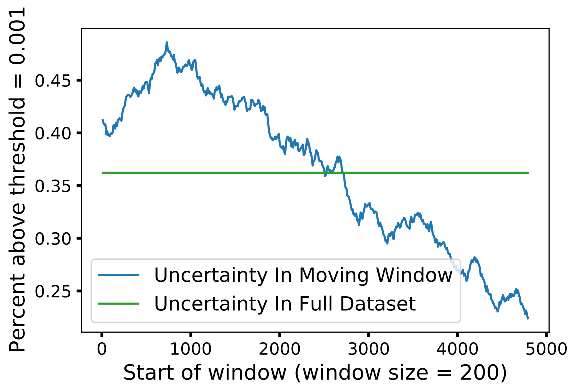
We proposed AdaCore, a method that leverages the topology of the dataset to extract salient subsets of large datasets for efficient machine learning. The key idea behind AdaCore is to dynamically incorporate the curvature and gradient of the loss function via an adaptive estimate of the Hessian to select weighted subsets (coresets) which closely approximate the preconditioned gradient of the full dataset. We proved an exponential convergence rate for first and second-order optimization methods applied to AdaCore coresets, under certain assumptions. Our extensive experiments, using various optimizers e.g., SGD, Ada-Hessian, and Newton's method, show that AdaCore can extract higher quality coresets compared to baselines, rejecting potentially redundant data points. This speeds up the training of various machine learning models, such as logistic regression and neural networks, by over 4.5x while selecting fewer but more diverse data points for training.

4.6.1 Further Research

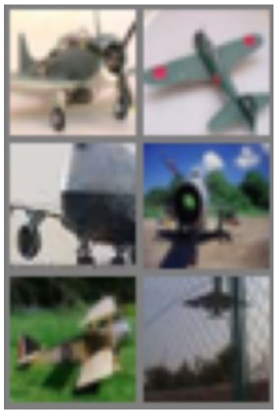
Currently AdaCore uses the Hutchinson method to estimate the diagonal of the Hessian matrix. One simple way to improve the curvature estimate is to use the general-purpose preconditioner from PSGD. This improved preconditioner should improve the subset selection quality of AdaCore.



(a) Forgetting vs class ranking



(b) Uncertainty vs class ranking


(c) Most selected

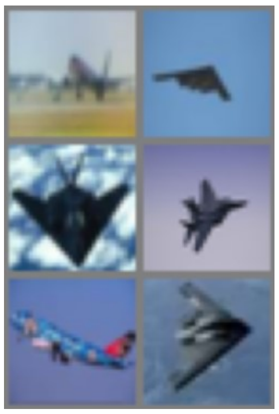

(d) Not selected

Figure 4.5: Training ResNet20 on S=1% subsets of CIFAR-10 selected by AdaCore. (a) Forgetting scores, and (b) uncertainty of examples in a class sorted by AdaCore at the end of training. AdaCore prioritize selecting more forgettable and uncertain examples. (c) Six images selected by AdaCore most frequently (25 times) from the airplane class. (d) The subset of images never selected by AdaCore.

CHAPTER 5

Generating High Fidelity Synthetic Data via Coreset selection and Entropic Regularization

Generative models have the ability to synthesize data points drawn from the data distribution, however, not all generated samples are high quality. In this chapter, we propose using a combination of coresets selection methods and "entropic regularization" to select the highest fidelity samples. We leverage an energy-based model which resembles a variational auto-encoder with an inference and generator model for which the latent prior is complexified by an energy-based model. In a semi-supervised learning scenario, we show that augmenting the labeled data-set, by adding our selected subset of samples, leads to better accuracy improvement rather than using all the synthetic samples.

5.1 Introduction

In machine learning, augmenting data sets with synthetic data has become a common practice that potentially provides significant improvements in downstream tasks such as classification. For example, in the case of images, recent methods like MixMatch, FixMatch, and Mean Teacher (BCG19) (Raf20) (TV17) have proposed data augmentation techniques that rely on simple pre-defined transformations such as cropping, resizing, etc.

However, generating augmentations is not as straightforward in all modalities. Hence, one suggestion is to use samples from generative models to augment the data sets. One issue that arises is that simply augmenting a data set using a generative model can often lead to the degradation of classification accuracy due to some poor samples drawn from the generator. The question arises: can we filter the lower-quality generated samples to avoid degradation in accuracy? In our method, we select a subset of synthetic samples which have high fidelity to the underlying data set via Craig (MBL19); additionally, we introduce "entropic regularization" by filtering samples with low entropy over the latent classifier.

In semi-supervised learning, the goal is to learn a classifier model which maintains high classification accuracy while reducing the number of labeled observed examples. Generative modeling, especially likelihood-based learning, is a principled formulation for unsupervised and semi-supervised learning. Within this family of models, energy-based models (EBM) are particularly convenient for semi-supervised learning, as they may be interpreted as generative classifiers. We have access to the class predictions and may also draw samples from the model.

Another direction in supervised learning is reducing the computation involved in training a model by reducing the data set to a smaller subset. Such sets are coined *coresets* as a smaller set of representative points that attempts to approximate the geometry of a larger point set under some metric. Recent art (MBL19) introduces a novel algorithm Craig which constructs a weighted coreset such that the gradient over the entire training data-set is closely estimated, which allows for gradient descent on the smaller coreset with considerable improvement in the sample- and computational-efficiency.

In this chapter, we show that semi-supervised learning and coreset subset selection are complementary and improve generalization and generation quality. First, a generative classifier is learned on a large set of unlabelled data and a small set of labeled data pairs. Then, the generative model is utilized to draw class-conditional samples, which augment the labeled data pairs. As such augmentation might be a considerably large set, we can draw infinite samples from the generative model; we recruit Craig to reduce the conditional samples to a much smaller coreset while approximately maintaining the full gradient over the cross-entropy term. As the generative model might synthesize conditional samples of low quality or even incorrect class identity, we apply an entropic filter to remove noisy samples. By learning a joint generative classifier, we learn a generator that can produce samples that improve classification accuracy and a classifier that can boost generative capacity and quality.

This method may be interpreted as a learned (and filtered) data augmentation as opposed to classical data augmentation in which the set of augmentation functions (e.g., convolution with Gaussian noise, horizontal or vertical flipping, etc.) is pre-defined and could be specific to a data-set or modality. We demonstrate the method's efficacy through a significant improvement in classification performance.

5.2 Related Work

Data augmentation. Semi-supervised models with purely discriminative learning mostly rely on data augmentation, exploiting the class-invariance properties of images. Pseudo-labels (Lee13) train a discriminative classifier on a small set of labeled data and sample labels for a large set of unlabelled data, which in turn is used to train the classifier further supervised. MixMatch (BCG19) applies stochastic transformations to an unlabeled image, and each augmented image is fed to a classifier for which the average logit distribution is sharpened by lowering the soft-max temperature. FixMatch (Raf20) strongly distorts an unlabeled image and trains the model such that the cross-entropy between the one-hot pseudo-labels of the original image and the logits of the distorted image is minimized. Mean teacher (TV17) employs a teacher model whose parameters are the running mean of a student model and trains the student such that a discrepancy between teacher and student predictions of augmented unlabeled examples is minimized. Virtual Adversarial Training (VAT) (MMK18) finds an adversarial augmentation to an unlabeled example within an ϵ-ball with respect to some norm such that the distance between the class distribution conditional on the unlabeled example and the one on the adversarial example is maximized.

The methods of MixMatch, FixMatch, and Mean teacher rely on pre-defined data augmentations, which are readily available in the modality of images as the semantic meaning is invariant to transforms such as rotation or flipping but are challenging to construct in modalities such as language or audio modalities. Our method is agnostic to the data modality. Pseudo-labeling is closely related in that labels are sampled given unlabeled examples, whereas our method samples examples given labels. VAT is close to our method as it is modality agnostic and leverages the learned model to sample labeled examples, albeit of an "adversarial" nature. In contrast, our samples are "complementary." DAPPER is closest to our method as it employs a generative model to augment the data set but misses the coreset reduction.

5.3 Synthetic Data Generation for Semi-Supervised Learning

Notation Let $x \in \mathcal{R}^D$ be an observed example. Let y be a K-dimensional one-hot vector as the label for classification with K categories. Suppose $\mathcal{L} = \{(x_i, y_i) \in \mathbb{R}^D \times \{k\}_{k=1}^K, i = 1, ..., M\}$ denotes a set of labeled examples where K indicates the number of categories and $\mathcal{U} = \{x_i \in \mathbb{R}^D, i = M+1, ..., M+N\}$ denotes a set of unlabeled examples.

Semi-Supervised Learning Let $p_\theta(y \mid x)$ denote a soft-max classifier with parameters θ. Semi-supervised learning aims to learn θ with "good" generalization while decreasing the number of labeled examples M.

5.3.1 Latent Energy Based Model

Let $z \in \mathbb{R}^d$ be the latent variables, where $D \gg d$. We assume a Markov chain $y \rightarrow z \rightarrow x$. Then the joint distribution of (y, z, x) is

$$p_\theta(y, z, x) = p_\alpha(y, z)\ p_\beta(x|z), \tag{5.1}$$

where $p_\alpha(y,z)$ is the prior model with parameters α, $p_\beta(x|z)$ is the top-down generation model with parameters β, and $\theta = (\alpha, \beta)$. Then, the prior model $p_\alpha(y,z)$ is formulated as an energy-based model (PHN20),

$$p_\alpha(y,z) = Z(\alpha)^{-1} \exp(F_\alpha(z)[y])\; p_0(z). \tag{5.2}$$

$p_0(z)$ is a reference distribution, assumed to be isotropic Gaussian. $F_\alpha(z) \in \mathbb{R}^K$ is parameterized by a multi-layer perceptron. $F_\alpha(z)[y]$ is the y_{th} element of $F_\alpha(z)$, indicating the conditional negative energy. $Z(\alpha)$ is the partition function. In the case where the label y is unknown, the prior model $p_\alpha(z) = \sum_y p_\alpha(y,z) = Z(\alpha)^{-1} \sum_y \exp(F_\alpha(z)[y]) p_0(z)$. Taking the log of both sides:

$$\log p_\alpha(z) = \log \sum_y \exp(F_\alpha(z)[y]) + \log p_0(z) - \log Z(\alpha), \tag{5.3}$$

Figure 5.1 illustrates the extension of the symbol-vector coupling to the latent-space energy-based model.

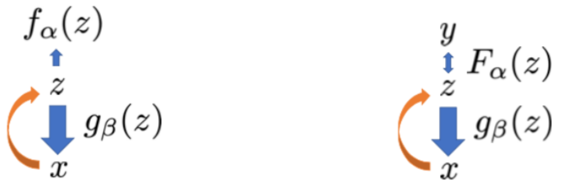


Figure 5.1: Lifting log-sum-exp into latent space. Left: Latent-space EBM f_α on top of generator g_β. Right: F_α allows mapping from label y to latent z and vice versa.

The prior model can be interpreted as an energy-based correction or exponential tilting of the reference distribution, p_0. The correction term is $F_\alpha(z)[y]$ conditional on y, while it is $\log \sum_y \exp(F_\alpha(z)[y])$ when y is unknown. Denote

$$f_\alpha(z) = \log \sum_y \exp(F_\alpha(z)[y]), \tag{5.4}$$

and then $-f_\alpha(z)$ is the free energy (GWJ19). The soft-max classifier is $p_\alpha(y|z) \propto \exp(\langle y, F_\alpha(z)\rangle) = \exp(F_\alpha(z)[y])$.

The generation model is the same as the top-down network in VAE (KW13b), $x = g_\beta(z) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2 I_D)$, so that $p_\beta(x|z) \sim \mathcal{N}(g_\beta(z), \sigma^2 I_D)$.

We use variational inference to learn our latent space EBM by minimizing the evidence lower bound (ELBO) over our energy, encoder, and generator models jointly. Refer to appendix 5.4 for more details about learning the model.

In summary, we can use the above model to i) classify data points, ii) generate class-conditional samples and iii) compute entropy for each generated sample. We will leverage these properties in the later sections to get better augmentation for our data set.

5.4 Learning the model with variational inference

Given a data point in the unlabeled set, $x \in \mathcal{U}$, the log-likelihood $\log p_\theta(x)$ is lower bounded by the evidence lower bound (ELBO),

$$\text{ELBO}(\theta) = \text{E}_{q_\phi(z|x)}[\log p_\beta(x|z)] - D_{KL}[q_\phi(z|x) \| p_\alpha(z)] \tag{5.5}$$

where $\theta = \{\alpha, \beta, \phi\}$ is overloaded for simplicity and $q_\phi(z|x)$, is a variational posterior, an approximation to the intractable true posterior $p_\theta(z|x)$.

For the prior model, the learning gradient, for example, is

$$\nabla_\alpha \text{ELBO}(\theta) = \text{E}_{q_\phi(z|x)}[\nabla_\alpha f_\alpha(z)] - \text{E}_{p_\alpha(z)}[\nabla_\alpha f_\alpha(z)] \tag{5.6}$$

where $f_\alpha(z)$ is the negative free energy defined in equation (5.4), $\text{E}_{q_\phi(z|x)}$ is approximated by samples from the variational posterior and $\text{E}_{p_\alpha(z)}$ is approximated with short-run MCMC chains (NPH20) initialized from the variational posterior $q_\phi(z|x)$.

Let $\psi = \{\beta, \phi\}$ collect parameters of the inference and generation models, and the learning gradients for the two models are,

$$\nabla_\psi \text{ELBO}(\theta) = \nabla_\psi \text{E}_{q_\phi(z|x)}[\log p_\beta(x|z)] - \nabla_\psi D_{KL}[q_\phi(z|x) \| p_0(z)] + \nabla_\psi \text{E}_{q_{\phi(z|x)}} f_\alpha(z) \quad (5.7)$$

where $D_{KL}[q_\phi(z|x)\|p_0(z)]$ is tractable, and the expectation in the other two terms is approximated by samples from the variational posterior distribution $q_\phi(z|x)$.

For one example of labeled data, $(x, y) \in \mathcal{L}$, the log-likelihood can be decomposed as $\log p_\theta(x, y) = \log p_\theta(x) + \log p_\theta(y|x)$. While we optimize $\log p_\theta(x)$ as the unlabeled data, we maximize $\log p_\theta(y|x)$ by minimizing the cross-entropy as in standard classifier training. Notice that given the Markov chain assumption $y \to z \to x$, we have

$$p_\theta(y|x) = \int p_\theta(y|z) p_\theta(z|x) dz = \text{E}_{p_\theta(z|x)} p_\theta(y|z) \approx \text{E}_{q_\phi(z|x)} \frac{\exp(F_\alpha(x)[y])}{\sum_k \exp(F_\alpha(x)[k])}. \quad (5.8)$$

In the last step, the true posterior $p_\theta(z|x)$, which requires expensive MCMC, is approximated by the amortized inference $q_\phi(z|x)$.

5.4.1 Sampling Synthetic data from the EBM

Naturally, increasing the cardinality of the set of labeled samples $\mathcal{L}$ may improve the classification accuracy of soft-max classifier $p_\theta(y \mid x)$. In the case of image models, traditional methods recruit a set of transformations or permutations of x, such as convolution with Gaussian noise or random flipping. Instead, we leverage the learned top-down generator $p_\beta(x|z)$ to augment $\mathcal{L}$ with class conditional samples. This is beneficial as (1) the generative path is readily available as an auxiliary model of learning the variational posterior $q_\phi(z|x)$ by auto-encoding variational Bayes, (2) hand-crafting of data augmentation is domain and modality-specific, and (3) in principle the number of conditional ancestral samples is infinite and might capture the underlying data distribution well.

We may construct the augmented set of L labeled samples $\mathcal{L}^+ = \{(x_i, y_i)\}$ by drawing conditional latent samples from the energy-based prior model $p_\alpha(y, z)$ in the form of Markov chains. Then, we obtain data space samples by sampling from the generator $p_\beta(x|z)$.

First, for each label y, we draw an equal number of samples $\mathcal{Z} = \{z_i\}$ in latent space. One convenient MCMC is the overdamped Langevin dynamics, which we run for T_{LD} steps with target distribution $p_\alpha(y, z)$,

$$z \sim p_0(z), \tag{5.9}$$

$$z_{t+1} = z_t + s\nabla_z \left[f_\alpha(z)[y] - \|z\|^2/\, 2\right] + \sqrt{2s}\epsilon_t,\ t = 1, \ldots, T_{LD} \tag{5.10}$$

with negative conditional energy $f_\alpha(z)[y]$, discretization step size s, and isotropic $\epsilon_t \sim N(0, I)$.

Then, we draw conditional samples $\{x_i\}$ in data space given $\{z_i\}$ from the top-down generator model $p_\beta(x|z)$,

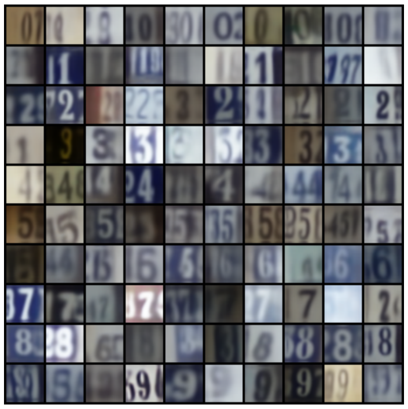
$$\mathcal{L}^+ = \{(x_i \sim p_\beta(x|z_i), y_i) \mid i = M + N, \ldots, M + N + L\} \tag{5.11}$$

which results in an augmented data set of L class conditional samples.

5.4.2 Entropic Regularization

When learning the generative classifier on both labeled samples $\mathcal{L}$ and the above naive construction of augmentation $\mathcal{L}^+$, the classification accuracy tends to be worse than solely learning from $\mathcal{L}$. As depicted in Figure 5.2a, a few conditional samples suffer from either low visual fidelity or even incorrect label identity. This reveals the implicit assumption of our method is that $\int p_\beta(x|z)p_\alpha(z|y)$ is reasonably "close" to the true class conditional distribution $p(x|y)$ under some measure of divergence, which is not guaranteed.

To address the issue of outliers, we propose to exclude conditional samples for which the


(a) Unsorted Conditional Samples.

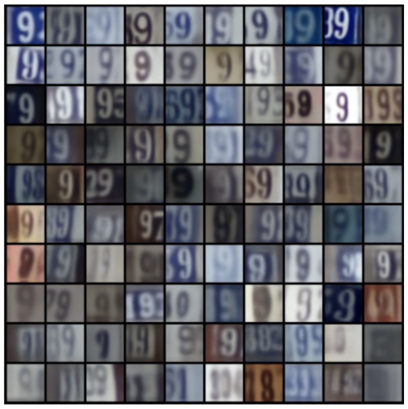

(b) Sorted Conditional Samples of Class 9.

Figure 5.2: Class conditional samples drawn from $p_\beta(x|z)p_\alpha(z|y)$. (a) Outliers suffer from low visual fidelity (e.g., the last sample in the row of "ones") or wrong label identity (e.g., the last image of the row of "sevenths". (b) Conditional samples sorted by increasing Shannon entropy $\mathcal{H}(z)$ over the logits.

entropy in logits $\mathcal{H}(p_\theta(y|z))$ exceeds some threshold $\mathcal{T}$. We propose the following two criteria for outlier detection,

$$\mathcal{H}_1(z) = \log \sum_y \exp(F_\alpha(z)[y]), \tag{5.12}$$

$$\mathcal{H}_2(z) = -\sum_y p_\theta(z|y) \log p_\theta(z|y). \tag{5.13}$$

Note,(5.13) is the classical Shannon entropy over the soft-max normalized logits of the classifier. Then, we may construct a more faithful data augmentation as follows,

$$\mathcal{Z}_\mathcal{T} = \{z_i \sim p(z|y_i) \mid \mathcal{H}(z_i) < \mathcal{T}, i = M+N, \ldots, M+N+L\}, \tag{5.14}$$

$$\mathcal{L}^+_\mathcal{H} = \{(x_i \sim p_\beta(x|z_i), y_i) \mid i = M+N, \ldots, M+N+L\}. \tag{5.15}$$

Figure 5.2b depicts conditional samples sorted by $\mathcal{H}(z)$ for which samples with relatively large Shannon entropy suffer from low visual fidelity.

The learning and sampling algorithm is described in Algorithm 2 as an extension of (PHN20).

Algorithm 2 Semi-supervised learning of generative classifier with coreset selection.

```
input  : Learning iterations $T$, augmentation iteration $T_a$, learning rates $(\eta_0, \eta_1, \eta_2, \eta_3)$, initial parameters $(\alpha_0, \beta_0, \phi_0)$, observed
         unlabelled examples $\{x_i\}_{i=1}^{M}$, observed labelled examples $\{(x_i, y_i)\}_{i=M+1}^{M+N}$, unlabelled, labelled and augmented batch
         sizes $(n, m, l)$, number of augmented samples $L$, entropy threshold $\mathcal{T}$, and number of Langevin dynamics steps $T_{LD}$.
output: $(\alpha_T, \beta_T, \phi_T)$.
1 for $t = 0 : T - 1$ do
2     1. Mini-batch:
      Sample $\{x_i\}_{i=1}^{m} \subset \mathcal{U}$, $\{x_i, y_i\}_{i=m+1}^{m+n} \subset \mathcal{L}$, and $\{x_i, y_i\}_{i=m+n+1}^{m+n+l} \subset \mathcal{L}_{\mathcal{H}}^{+}$.
      2. Prior sampling:
      For each unlabelled $x_i$, initialize a Markov chain $z_{0_i}^{-} \sim q_\phi(z|x_i)$ and update by MCMC with target distribution $p_\alpha(z)$ for
      $T_{LD}$ steps.
      3. Posterior sampling:
      For each $x_i$, sample $z_i^{+} \sim q_\phi(z|x_i)$ using the inference network and reparameterization trick.
      4. Unsupervised learning of prior model:
      $\alpha_{t+1} = \alpha_t + \eta_0 \frac{1}{m} \sum_{i=1}^{m} [\nabla_\alpha F_{\alpha_t}(z_i^{+}) - \nabla_\alpha F_{\alpha_t}(z_i^{-})]$.
      5. Unsupervised learning of inference and generator models:
      $\psi_{t+1} = \psi_t + \eta_1 \frac{1}{m} \sum_{i=1}^{m} [\nabla_\psi [\log p_{\beta_t}(x|z_i^{+})] - \nabla_\psi \mathrm{KL}(q_{\phi_t}(z|x_i) \| p_0(z)) + \nabla_\psi [F_{\alpha_t}(z_i^{+})]$.
      6. Supervised learning of prior and inference model:
      $\theta_{t+1} = \theta_t + \eta_2 \frac{1}{n} \sum_{i=m+1}^{m+n} \sum_{k=1}^{K} y_{i,k} \log(p_{\theta_t}(y_{i,k}|z_i^{+}))$.
      7. Augment at iteration $T_a$:
      $\mathcal{Z}_{\mathcal{T}} = \{z_i \sim p(z|y_i) \mid \mathcal{H}(z_i) < \mathcal{T}, i = M + N, \ldots, M + N + L\}$,
      $\mathcal{L}_{\mathcal{H}}^{+} = \{(x_i \sim p_\beta(x|z_i), y_i) \mid i = M + N, \ldots, M + N + L\}$.
      8. Approximate the gradient below with CRAIG after iteration $T_a$ according to (5.17):
      $\theta_{t+1} = \theta_{t+1} + \eta_3 \frac{1}{n} \sum_{i=n+m+1}^{m+n+l} \sum_{k=1}^{K} y_{i,k} \log(p_{\theta_t}(y_{i,k}|z_i^{+}))$.
3 end for
```

5.4.3 Coreset Selection

Training machine learning models on large data sets incurs considerable computational costs. There has been substantial effort to develop subset selection methods that can carefully select a subset of the training samples that generalize on par with the entire training data (MBL19) (PDM22b). Since we can generate a virtually infinite amount of synthetic samples, we must select the best subset of points to augment our base data set with. Intuitively CRAIG selects a subset that can best cover the gradient space of the full data set. It does this by selecting exemplar medoids from clusters of data points in the gradient space. As a bi-product, CRAIG robustly rejects noisy and even poisoned data points. The subset corset algorithm ADACORE improves on CRAIG's results by selecting diverse subsets (PDM22b). Utilizing coreset methods allows us to select samples from the generator representative of the ground truth data set while rejecting points that may negatively impact our network performance.

Formally, the CRAIG (MBL19) algorithm aims to identify the smallest subset $S \subset V$ and corresponding per-element stepsizes $\gamma_j > 0$ that approximate the full gradient with an error at most $\epsilon > 0$ for all the possible values of the optimization parameters $w \in \mathcal{W}$.

$$S^* = \arg \min_{S \subseteq V, \gamma_j \geq 0 \forall j} |S|, \text{ s.t. } \quad \max_{w \in \mathcal{W}} \left\| \sum_{i \in V} \nabla f_i(w) - \sum_{j \in S} \gamma_j \nabla f_j(w) \right\| \leq \epsilon \tag{5.16}$$

For deep neural networks, calculating the above metric is more costly than calculating vanilla SGD. In deep neural networks, the variation of the gradient norms is mostly captured by the gradient of the loss w.r.t the inputs of the last layer L. (MBL19) shows that the normed gradient difference between data points can be efficiently bounded approximately by

$$\|\nabla f_i(w) - \nabla f_j(w)\| \leq c_1 \left\| \Sigma'_L \left(z_i^{(L)}\right) \nabla f_i^{(L)}(w) - \Sigma'_L \left(z_j^{(L)}\right) \nabla f_j^{(L)}(w) \right\| + c_2 \tag{5.17}$$

where $z_i^{(l)} = w^{(l)} x_i^{(l-1)}$. This upper bound is only slightly more expensive than calculating the loss. In the case of cross entropy loss with soft-max as the last layer, the gradient of the loss w.r.t. the i-th input of the soft-max is $p_i - y_i$, where p_i are logits and y is the one-hot encoded label. As such, in this case, Craig does not need a backward pass or extra storage. This makes Craig a practical and scalable tool to select high-quality generated synthetic data points.

5.4.4 Implicit learned data augmentation

In the following, we will re-interpret the above explicit data augmentation and entropic regularization into an implicit augmentation which can be merged into a simple term of the learning objective function.

The assumed Markov chain underlying the model is $y \to z \to x$. Let $\hat{z} \sim q_\phi(z|x)$ denote the conditional sample $\hat{z}$ from the approximate posterior given an observation x. Let $\hat{y} \sim p_\theta(y|\hat{z})$ denote the predicted label for which the logits of C classes are given as $F_\alpha(z) = (F_\alpha(z)[1], F_\alpha(z)[2], \ldots, F_\alpha(z)[C])$.

The factorization which recruits the log-sum-exp lifting (5.3) as exponential tilting of the reference distribution $p_0(z)$ so that the conditional $p_\alpha(y|z)$ is defined and amortized inference (7.6) with a variational approximation of the posterior $q_\phi(z|x)$. These conditional distributions allow us to express learned data augmentation as the chain,

$$y \overset{q_T(z|y)}{\to} z \overset{p_\theta(x|z)}{\to} x \overset{q_\phi(z|x)}{\to} \hat{z} \overset{F_\alpha(z)[y]}{\to} \hat{y}. \tag{5.18}$$

in which the conditional $z|y$ is given as an MCMC dynamics. Specifically, we define $q_T(z|y)$ as K-steps of an overdamped Langevin dynamics on the learned energy-based prior $\exp(F_\alpha(z)[y])p_0(z)$, which iterates

$$z_{k+1} = z_k + s\nabla_z \log p(z_k|y) + \sqrt{2Ts}\epsilon_k, \ \ k = 0, \dots, K-1, \tag{5.19}$$

with discretization step-size s, temperature T and isotropic noise $\epsilon_k \sim N(0, I)$.

For the (labeled) data distribution p_{data} the labels y are known. For the data augmentation, we assume a discrete uniform distribution over labels $y \sim U\{1, C\}$. Then, we define augmentation of synthesized examples as the marginal distribution

$$p_{aug}(x) = E_y E_{z|y}[p(x|z)p(z|y)]. \tag{5.20}$$

Then, we may introduce an augmented data distribution as the mixture of the underlying labeled data distribution p_{data} and the augmentation p_{aug} and mixture coefficient λ,

$$p_\lambda(x) = \lambda p_{\text{data}}(x) + (1-\lambda)p_{\text{aug}}(x). \tag{5.21}$$

As we have access to $p_\theta(y|x) = E_{p_\theta(z|x)} p_\theta(y|z)$ and can extend the objective to minimize the KL divergence under the augmented data distribution such that the labels y of (labeled)

p_{data} and p_{aug} are recovered under the model,

$$E_{p_\lambda(x)}[KL(p(y|x)\|p(\hat{y}|x))]. \tag{5.22}$$

In information theory, the Kraft-McMillian theorem relates the relative entropy $KL(p\|q) = E_p[\log p/q]$ to the Shannon entropy $H(p)$ and cross-entropy $H(p, q)$,

$$KL(p\|q) = H(p, q) - H(p). \tag{5.23}$$

In our case, the first term reduces to soft-max cross entropy over the (labeled) data distribution p_{data} and sampled labels $y \sim U\{1, C\}$. Hence, to minimize the above divergence, we must minimize the cross entropy, which is consistent with classical learning of discriminative models. However, note that in our case the steps in (5.18) are fully differentiable so that the data augmentation turns into an implicit term in the unified objective function rather than an explicitly constructed set of examples.

Lastly, we wish to re-introduce the entropic regularization for implicit data augmentation. Note, the entropic filter can be interpreted as a hard threshold on $H(p(\hat{y}|x))) < \mathcal{T}$. Here the Langevin dynamics q_T on z maximizes the logit $F_\alpha(z)[y]$, i.e. minimizes $H(p(\hat{y}|x)))$, for which the Wiener process materialized in the noise term $\sqrt{2Ts}\epsilon_k$ with temperature T introduces randomness, or, smoothens the energy potential such that the dynamics converge towards the correct stationary distribution. High temperature T leads to Brownian motion, while low T leads to gradient descent. We realize that T controls $H(p(\hat{y}|x)))$ as it may be interpreted as a soft or stochastic relaxation of $\mathcal{T}$. We can express the entropic filter in terms of the temperature T of q_T and only need to lower T to obtain synthesized samples with associated low entropy in the class logits.

5.5 Experiments: Learning data augmentation

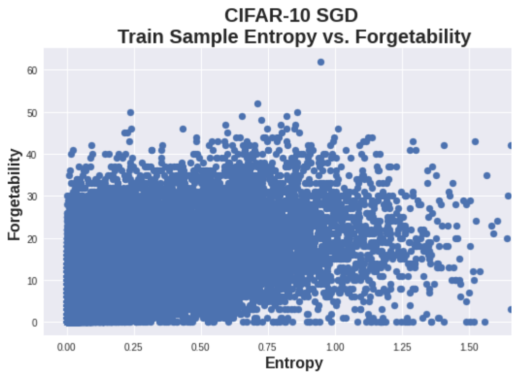
We evaluate our method on standard semi-supervised learning benchmarks for image data. Specifically, we use the street view house numbers (SVHN) (NWC11) data set with $1,000$ labeled images and $64,932$ unlabeled images. The inference network is a standard Wide ResNet (ZK16). The generator network is a standard 4-layer de-convolutional network regularly used in DC-GAN. The energy-based model is a fully connected network with 3 layers. Adam (KB14) is adopted for optimization with batch sizes $n = m = l = 100$. The models are trained for $T = 1,200,000$ steps with augmentation after $T_a = 600,000$ steps. The short-run MCMC dynamics in (7.12) is run for $T_{LD} = 60$ steps.

At iteration T_a, we take L class conditional samples from the generator with an equal amount of samples ($L/10$ for each digit). We filter conditional samples based on $\mathcal{H}$ as described in Section 5.4.2 for which the threshold $\mathcal{T} = 1\text{e}{-6}$ was determined by grid search. Next, we run Craig on the generated samples to keep a subset of 10% of the samples. For these additional examples, we compute the soft-max cross-entropy gradient with per-example weights obtained by Craig and update the model with step size $\eta_3 < \eta_2$ or a loss coefficient of 0.1 to weaken the gradient of $\mathcal{L}^+_{\mathcal{H}}$ relative to the original labeled data $\mathcal{L}$. Additionally, for every 10k iteration, we rerun Craig to choose an updated subset of generated samples.

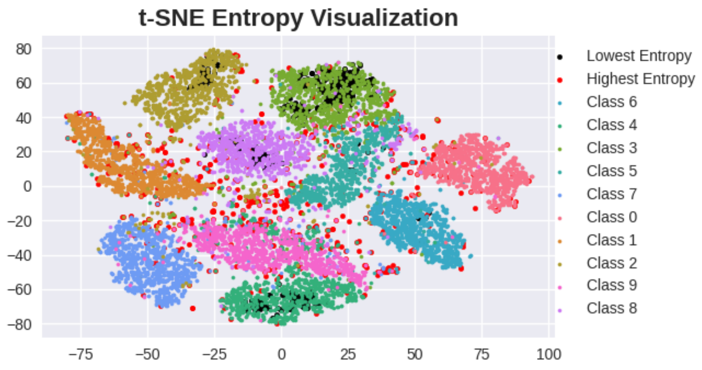
			L		
Method	0	10,000	40,000	100,000	200,000
Baseline	92.0 ± 0.1	88.1 ± 0.1	-	-	-
$\mathcal{H}$	-	93.5 ± 0.1	93.8 ± 0.1	-	-
$\mathcal{H}$ & Craig	-	93.0 ± 0.1	93.5 ± 0.1	93.9 ± 0.1	93.9± 0.1
$\mathcal{H}$ & Craig & PL	-	-	**94.5 ± 0.1**	-	-

Table 5.1: Test accuracy with varied number of conditional samples L on SVHN (NWC11).

Table 5.1 depicts results for the test accuracy on SVHN for a varied number of conditional samples L. First, we learned the model without data augmentation as a baseline. Then,



(a) Forgettability vs Entropy



(b) t-SNE Entropy Visualization

Figure 5.3: a) Shows that points that have low entropy are correlated with unimportant forgettable points b) Shows that central cluster points correspond to low entropy points.

we draw L conditional samples without an entropic filter and observe worse classification performance. As described earlier, we introduce the entropic filter $\mathcal{H}$ to eliminate conditional samples of low quality, which significantly improves classification performance with increasing L. Finally, we combine the entropic filter $\mathcal{H}$ and coreset selection by CRAIG to increase L further. For $L = 10,000$, there is a significant improvement in classification accuracy when introducing CRAIG, which, however, decreases with increasing L. Lastly, to further boost accuracy, we pseudo-label unlabeled data points from the SVHN data set using the latent classifier. We reject data points whose entropy over the latent classifier is above 10^{-6}. See 5.2 for more results.

5.6 Comparison to other Methods

Here we include some different methods in Table 5.2. Our model outperforms FlowGMM and JEM, and other likelihood-based models. The improvement is especially clear on SVHN (with almost 10% absolute improvement compared to FlowGMM). Furthermore, we close the performance gap between our model and GAN-based and discriminative methods, which are highly tuned for images as we beat TrippleGAN.

All competing methods use explicit image-based data transformations to augment their

			L		
Method	0	10,000	40,000	100,000	200,000
Baseline	92.0 $\pm$ 0.1	88.1 $\pm$ 0.1	-	-	-
$\mathcal{H}$	-	93.5 $\pm$ 0.1	93.8 $\pm$ 0.1	-	-
$\mathcal{H}$ & CRAIG	-	93.0 $\pm$ 0.1	93.5 $\pm$ 0.1	93.9 $\pm$ 0.1	93.9$\pm$ 0.1
$\mathcal{H}$ & CRAIG & PL	-	-	**94.5 $\pm$ 0.1**	-	-
VAE M1+M2	64.0 $\pm$ 0.1				
AAE	82.3 $\pm$ 0.3				
JEM	66.0 $\pm$ 0.7				
FlowGMM	82.4				
TripleGAN	94.2 $\pm$ 0.2				
BadGAN	95.6 $\pm$ 0.03				
Π-Model	94.6 $\pm$ 0.2				
VAT	96.3 $\pm$ 0.1				

Table 5.2: Test accuracy with varied number of conditional samples L on SVHN (NWC11).

base dataset. Instead, we learn augmentations that can be applied to domains that do not have explicit augmentation transforms, such as audio, text, etc.

Note the coreset method can be used to refine samples of competing generative methods.

5.6.0.1 Difficult Examples hurt Contrastive Lethat arning

Toneva (TSC18) showed that points with high forgettability are the most critical for generalization in supervised learning. In **Chaper 3** Figure B.5, we showed that there is a strong correlation between points with high forgettability and high entropy. Furthermore, we showed in section B.1.4.1 Table B.3 that training on only the high entropy points leads to the same generalization as training on the full dataset and outperforms training on only the low entropy by 25.1%. Furthermore, when we plot gradient clusters from supervised learning and show where the high and low entropy points show up on the plot using t-SNE, we find that the core points of the clusters correspond to low entropy points. Yet in this chapter, we find that if we accept high entropy points from the generator, our contrastive model will degrade in performance in row 1 of Table 5.1. Effectively, the most critical subsets for SSL and contrastive learning are the least important for supervised learning.

Easy Examples are the Most Important Finally, to speed up training and select the most salient data points sampled from $p_\lambda(x)$to train on, we select a coreset of 10% of the generated dataset. In this, we see only a marginal degradation (between rows 2 and 3 of Table 5.1) to the classification accuracy while benefiting from significant speedup. This demonstrates that taking the representative core points is the most important for contrastive learning.

5.7 Conclusion

In the semi-supervised learning setting, we have investigated combining generative models with a coreset selection algorithm, Craig. Such a combination is appealing as a generative model can, in theory, sample an infinite amount of labeled data. At the same time, a coreset algorithm can reduce such a large set to a much smaller informative set of synthesized examples. Moreover, learned augmentation is helpful as many discrete data modalities, such as text, audio, graphs, and molecules, do not allow the definition of hand-crafted semantically invariant augmentations (such as rotations for images) easily.

We illustrated that a naive implementation of this simple result deteriorates the classifier's accuracy over a baseline without such data augmentation. The underlying issue here was isolated to being related to the Shannon entropy in the predicted logits over classes for a synthesized example. High entropy indicates samples with low visual fidelity or wrong class identity, which may confuse the discriminative component of the model and lead to a loop in which uncertainty in the predictions leads to worse synthesis. In the first attempt, we constraint the class entropy in the set of augmented examples by taking a subset of the generated data set with a hard threshold on the Shannon entropy. This resulted in significant empirical improvement of classification accuracy of two percentage points on SVHN.

Moreover, we introduced pseudo labels, which further improved performance.

Then, we show that the latent energy-based model with symbol-vector couplings has conditional distributions for readily available end-to-end training of learned augmentations. We formulate learned data augmentation as the KL-divergence between two known conditional distributions, show the relation to cross-entropy, and relax the entropy regularization into the temperature of the associated Langevin dynamics. This allows learning data augmentations as an alteration of the learning objective function and paves the way toward a theoretical analysis.

Finally, in contrast to supervised learning, the most critical data points for self- and semi-supervised learning under contrastive settings are the low-entropy forgettable points.

One final note: when these experiments were run, AdaCore was still in its infancy. The results from **Chapter 4** show that AdaCore selects better quality subsets compared to Craig. As such, one would expect AdaCore to improve the results from this chapter.

CHAPTER 6

Gamma-VAE: Speech Re-Synthesis and Beyond

In this chapter, we study different Variational Autoencoders (VAEs) decoder distributions in the audio setting to see how to improve magnitude and phase reconstruction on speech resynthesis tasks. We first provide background on the existing decoder distributions, such as Complex Gaussian and Laplace, which are equivalent to a Gamma decoder under certain conditions. We then consider separately modeling speech's magnitude and phase information to see if we can improve the quality of either component, yielding an improvement in speech resynthesis. Extensive experiments show the Gamma decoder significantly improves magnitude reconstruction and that the von Mises decoder can weakly learn phase information. The novel Gamma decoder outperforms previous approaches, achieving a near-perfect PESQ of 4.4, representing a 42% improvement upon the state-of-the-art IS-VAE and an 86% decrease in the FAD metric. Our results demonstrate the effectiveness of the novel approach, improving the quality of speech resynthesis and compression capacity of VAEs.

6.1 Introduction

The goal of the VAE is to find an optimal trade-off between rate and distortion (APF), such that the reconstructed data is as close as possible to the original data while still requiring a minimal amount of information to represent it.

In the audio setting, VAEs have been applied to tasks such as music generation, speech synthesis, speech source-filter representation and audio denoising (FBD09; LAG19; GRH19;

BGL21; Nak20; WWR21). One key aspect of VAE performance is the choice of decoder loss function, which determines how well the model is able to capture the underlying distribution of the data. Different decoder distributions have been used in the context of VAEs for speech, including Gaussian decoders, complex-valued probability density functions, and a combination of magnitude and phase reconstruction (FBD09; LAG19; GRH19; BGL21; Nak20; WWR21). Yet, the Itakura-Saito (IS) VAE has emerged as the state-of-the-art (SOTA) decoder loss, thanks to its ability to accurately model the distribution of audio signals and generate high-quality samples.

This chapter considers the effect of different decoder distributions on audio-quality speech resynthesis, as well as compression capacity. We explore whether we can learn phase information in our decoder formulation and compare it to the IS-VAE. We believe that if we can find a better spectrum representation via a new decoder, it will improve current VAE-based audio processing methods that currently use IS-divergence.

To this end, we explore multiple novel decoder distributions that can more accurately capture the underlying distribution of speech signals. This results in improved reconstruction performance on speech resynthesis tasks. The role of the decoder distribution in the VAE architecture and its relationship to speech codec is discussed first. Our novel decoder distributions are then introduced, and results demonstrating their improved performance on speech resynthesis tasks compared to the IS-VAE are presented.

Our experimental findings suggest that: a) Phase information can be learned to a limited extent; b) There exist many distributions that can model speech leading to different properties; c) The Gamma distribution is the best-tested model for speech re-synthesis and compress-ability; d) Network architecture does not greatly affect performance. All models are evaluated using the Perceptual Evaluation of Speech Quality (PESQ) metric (RBH01), which ranges from [-0.5, 4.5], and the Fŕechet Audio Distance (FAD) metric (KZR18), whose range is unbounded, with a lower value indicating a greater similarity between the two sets of audio features. Unlike PESQ, FAD does not require a reference signal but instead takes the distance in the embedding space of an NN-based audio classifier between a distribution of natural audio and a generated dataset. The best-performing decoder explored in this chapter is the Gamma decoder. This decoder achieves a near-perfect PESQ of 4.4, improves PESQ by 42% and reduces FAD by 86% compared to the IS-VAE.

6.2 Background

In recent years, the Itakura-Saito VAE (IS-VAE) has become the SOTA VAE used in the domain of speech modeling. Typically, the IS-VAE models the probability density function (pdf) of the magnitude spectrum of the Short-Time Fourier Transform (STFT) domain. Only considering the magnitude spectrum of the speech signal, rather than both the magnitude and phase, can potentially lead to a loss of quality of the synthesized speech. To address this issue, (Nak20) proposed using a VAE to directly model the speech signal using a complex-valued Gaussian decoder. This approach may not be the most effective for capturing phase information because the magnitude and phase may follow different distributions. To this end, we propose modeling the magnitude and phase of speech separately using a positive distribution for the magnitude and a circular distribution for the phase. By inputting $x = [x_r, x_i]$, where $|x| = \sqrt{x_r^2 + x_i^2} = r$ represents the magnitude spectrum of the speech signal, and estimating both the magnitude, $|\hat{x}|$, and phase, $\angle x$, we aim to improve the performance of VAEs for speech re-synthesis tasks. This approach differs from the recently proposed method (Nak20), which uses a singular complex-valued Gaussian distribution to model the magnitude and phase of the signal x.

Overall, our proposed method offers a new approach to modeling the magnitude and phase of speech separately, with the goal of improving the performance of VAEs in speech re-synthesis tasks.

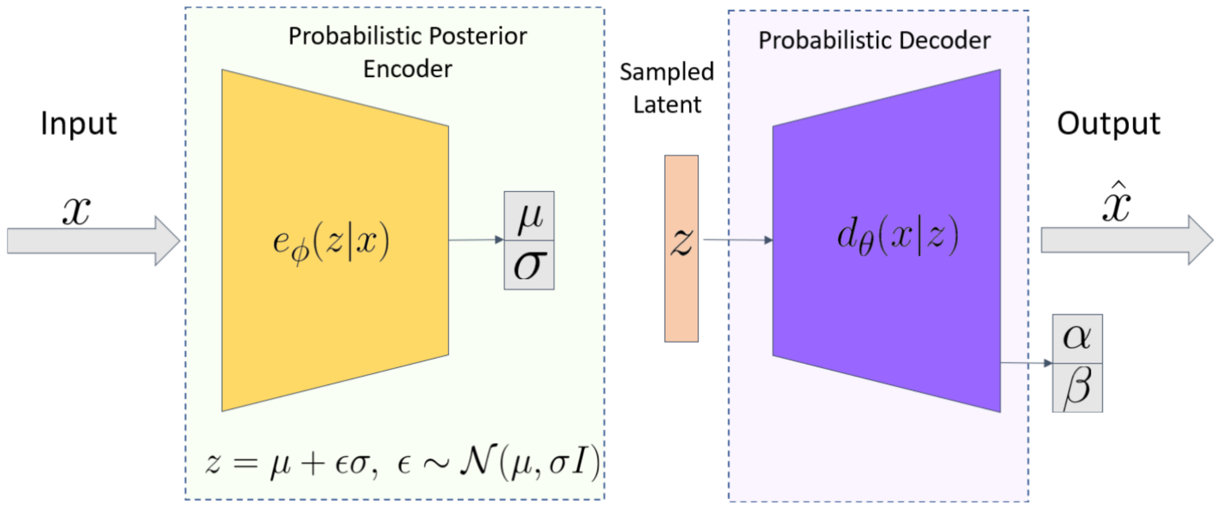


Figure 6.1: VAE: Where x is complex spectrogram of original signal and $\hat{x}$ is estimated via the parameters of the decoder distribution. $\hat{x}$ is complex or real depending on decoder distribution.

6.3 Problem Setting

The VAE is a framework that learns a latent representation, z, of the data x, which captures the underlying structure of the data distribution. VAEs are trained to learn an encoder function, $e_\phi(z|x)$, which maps the input data x to the latent space z. The approximate posterior distribution over the latent variables is given by $e_\phi(z|x)$. The encoder estimates the parameters of a normal distribution in the latent space, such as the mean and variance, which define the location and scale of the distribution. The VAE also learns a marginalized distribution, $m_\theta(z)$, for the latent variable encoding, and a decoder function, $d_\theta(x|z)$. Thus the joint distribution, $p_\theta(x, z)$, can accurately model the true data distribution, $p(x)$. The decoder estimates the parameters of a distribution, such as the shape and rate parameters of a Gamma distribution, to reconstruct the input signal x. The decoding side defines a joint pdf, $q_\theta(x, z) = m_\theta(z)d_\theta(x|z)$, with everything explicit, resulting in a joint pdf, $p_\phi(x, z) = p(x)e_\phi(z|x)$ that we can readily draw samples from, although $p(x)$ is unknown.

The VAE is based on the principle of maximum likelihood estimation (MLE), where the goal is to maximize the likelihood of the training data under the model. To make the optimization process more feasible, the VAE introduces the variational reparameterization trick, which involves re-parameterizing the random noise used to sample from the latent distribution so that the resulting samples can be transformed into the latent space in a differentiable manner. By using this trick, VAEs can be trained more efficiently on large datasets using a variational lower bound on the log-likelihood, written as:

$$\mathcal{L}(\theta, \phi; x) = \mathbb{E}_{e_\phi(z|x)} \left[\log d_\theta(x|z)\right] - \mathrm{KL}\left(e_\phi(z|x) || m_\theta(z)\right) \tag{6.1}$$

where x is a single training example, and θ and ϕ are the parameters of the decoder and encoder, respectively. The first term in the lower bound, $\mathbb{E}_{e_\phi(z|x)} \left[\log d_\theta(x|z)\right]$, is known as the reconstruction loss, and it measures the difference between the reconstructed data and the original data. The second term, $\mathrm{KL}\left(e_\phi(z|x) || m_\theta(z)\right)$, is known as the KL divergence, and it measures the difference between the approximate posterior distribution and the latent distribution.

6.4 Understanding Decoder Distributions

One way to understand the impact of phase information on VAEs for speech modeling is to consider a general decoder that models both the magnitude and phase of speech signals. There are two main categories of decoder distributions that can be used for this purpose: (Type I) decoders that directly model the complex-valued signal $\hat{x} = [\hat{x}_r, \hat{x}_i]$, such as Gaussian or Laplace decoders, and (Type II) Decoders that model the magnitude $|\hat{x}|$ and phase $\angle\hat{x}$ information separately, such as a Positive Distribution for magnitude and Circular Distribution for phase. The Itakura-Saito VAE (IS-VAE) can be seen as a special case of the general family of decoders that only model magnitude information. However, the general family of decoders has the potential to directly generate phase information, as they are theoretically phase-aware. Since there is a clear connection between magnitude and phase information in speech signals, modeling phase information may inherently improve the modeling of the magnitude spectrum. This is in line with the assumption often made by Vocoders (MG76), which is that the phase of a speech signal can be approximately reconstructed from its magnitude spectrum.

In this chapter, for Type II decoders we consider Log-Normal and Gamma distributions to model magnitude and von Mises distribution to model phase information of speech. The Log Normal Distribution corresponds to the Log Spectral Distance used in speech codec (RRJ93) and the Gamma distribution has been used as a strong prior in noise suppression (MIY14; DB19) and as we will show is directly related to the IS-divergence. The von Mises distribution is a natural choice for modeling the phase of speech because it is a circular continuous and smooth distribution, which allows for more accurate modeling of the phase information in speech signals. Additionally, it is symmetric around the mean, which is useful for modeling the cyclical nature of phase. See Table 6.1 for all decoder distributions used in this paper.

For all experiments, we consider the pdf of bins which are conditionally independent given z, in the STFT domain. We only consider one bin, $x = [x_r, x_i]$, and assume that power and phase spectra are conditionally independent given z.

6.4.1 Directly Modeling The Signal

Many decoder distributions exist that implicitly model the magnitude and phase components of speech. Here we discuss the Normal and Laplace Decoders.

Complex Normal The Gaussian decoder models a complex-valued spectrogram, implicitly modeling the magnitude & phase of speech

$$d_\theta(x|z) = \frac{1}{\pi\sigma^2} e^{-|x-\mu|^2/\sigma^2}. \tag{6.2}$$

Table 6.1: Formulations of Type I and Type II a & b decoders such as Complex Normal & Laplace, Log Normal von Mises, Gamma von Mises, Gamma, and Itakura-Saito. These decoder distributions can be used to derive negative log-likelehoods as done in eq. (6.9) & eq. (6.12).

Decoder Distribution	Formulation	Models
Complex Normal eq. (6.2)	$\frac{1}{\pi\sigma^2}e^{-\lvert x-\mu\rvert^2/\sigma^2}$	Joint Complex Spectrum
Complex Laplace eq. (6.3)	$\frac{1}{2\pi\sigma^2}e^{-\lvert x-\mu\rvert/\sigma}$	Joint Complex Spectrum
Log Normal von Mises	$\frac{\exp\left(\kappa\frac{x^\intercal x_0}{\lvert x\rvert\lvert x_0\rvert}-\frac{(\log\lvert x\rvert/\log\lvert x_0\rvert)^2}{2\sigma^2}\right)}{(2\pi)^{1.5}\,\sigma\,I_0(\kappa)\,\lvert x\rvert^2}$	Separate Magnitude & Phase
Gamma von Mises eq. (6.7)	$\frac{\beta^\alpha\lvert x\rvert^{\alpha-2}e^{\kappa\frac{x^\intercal x_0}{\lvert x\rvert\lvert x_0\rvert}-\beta\lvert x\rvert}}{2\pi\,I_0(\kappa)\,\Gamma(\alpha)}$	Separate Magnitude & Phase
Gamma eq. (6.10)	$\frac{e^{-\frac{\lvert x\rvert^2}{2\sigma^2}}}{2\pi\sigma^2}$	Magnitude Only
Itakura-Saito eq. (6.11)	$\frac{1}{\Gamma(\alpha)}\frac{\lvert x\rvert^\alpha}{\lvert y\rvert^\alpha}\lvert x\rvert^{-2}e^{-\frac{\lvert x\rvert}{\lvert y\rvert}}$	Magnitude Only

Complex Laplace Similarly, the Laplace decoder models a complex-valued spectrogram, modeling the magnitude & phase of speech

$$d_\theta(x|z) = \frac{1}{2\pi\sigma^2}e^{-|x-\mu|/\sigma}. \tag{6.3}$$

Assuming that $\mu = 0$, the probability density functions of both decoders are related to the Gamma distribution. Specifically, $|x|^2$ and $|x|$ follow the Gamma distribution for Gaussian and Laplace decoders, respectively. Empirical results demonstrate that for speech signals, $\mu \to 0$. However, it is not currently known if it is possible to learn μ theoretically.

Another similar decoder is the multivariate Laplace distribution. We forego this decoder since it does not correspond to either the Gamma or log-normal distribution even though it may have more favorable properties than the complex Laplace one.

6.4.2 Separately Modeling Magnitude and Phase

To explicitly capture both magnitude and phase information, we propose combining a positive distribution, such as the Log Normal or Gamma distribution, to capture magnitude information and a circular distribution, such as the von Mises distribution, which is an approximation to the Circular Normal distribution, to capture phase information.

Gamma + von Mises Decoder generates three outputs, α, β, and κ: α represents the shape parameter, β represents the rate parameter, and $\frac{\alpha}{\beta} = |\hat{x}|$ determines the central location of the Gamma distribution. The precision of the phase spectrum of the von Mises distribution is represented by κ. The probability density function of x can be expressed as a function of its parameters, which varies depending on the specific distribution. We use change of variables to rewrite the pdf $p(x_r, x_i)$ as $p(|x|, \angle x)$.

$$p(x_r, x_i) = \left|\tfrac{\partial(r,\theta)}{\partial(x_r,x_i)}\right|^{-1} p(r,\theta) = \frac{1}{|x|} p(|x|)p(\theta) \tag{6.4}$$

where

$$p(|x|) = \frac{\beta^\alpha |x|^{\alpha-1} e^{-\beta|x|}}{\Gamma(\alpha)}, \quad p(\theta) = \frac{e^{\kappa\cos(\theta-\theta_0)}}{2\pi I_0(\kappa)} \tag{6.5}$$

where Γ is a Gamma function, I_0 is a Bessel function, and $p(\theta)$ is the von Mises distribution. We can write the decoder as

$$d_\theta(x|z) = \frac{1}{|x|} \frac{\beta^\alpha |x|^{\alpha-1} e^{-\beta|x|}}{\Gamma(\alpha)} \frac{e^{\kappa \frac{x^\intercal x_0}{|x||x_0|}}}{2\pi I_0(\kappa)} \tag{6.6}$$

which can be rewritten with a more pleasant form as

$$d_\theta(x|z) = \frac{\beta^\alpha |x|^{\alpha-2} e^{\kappa \frac{x^\intercal x_0}{|x||x_0|} - \beta|x|}}{2\pi\, I_0(\kappa)\, \Gamma(\alpha)} \tag{6.7}$$

where α, β and κ are functions of z (all defined by the decoder). We constrain $\alpha > 0$ and $\beta > 0$ through reparameterization. As such we can formulate our loss function as the sum of the natural log of $d_\theta(x|z)$ and the KLD as in eq (6.1). The negative Log Loss becomes:

$$\begin{aligned}-\log d_\theta(x|z) = {} & \log 2\pi + \log I_0(\kappa) + \log\Gamma(\alpha) + \beta|x| \\ & -\alpha\log\beta - (\alpha-2)\log|x| - \kappa\frac{x^\intercal x_0}{|x||x_0|}\end{aligned} \tag{6.8}$$

When $\kappa \to 0$ the von Mises distribution becomes uniformly distributed and the phase information is not learned.

6.4.3 Phase Agnostic Decoders

Some decoders, such as the IS-VAE, do not model phase information. To generate a signal, one will need to use the ground truth oracle phase or find a magnitude consistent phase via Griffin-Lin style methods (GL83). This style of magnitude spectrum modeling is tantamount to having a von Mises distribution with infinite dispersion. As such we derive the Gamma distribution as a special case of the Gamma von Mises distribution, with $\kappa = 0$. We then derive the IS-VAE as a Gamma VAE.

Gamma Distribution Consider the Gamma von Mises Decoder which produces three outputs, α, β, and κ. Where α is the shape, β is the rate and $\alpha/\beta = |\hat{x}|$ gives the center location of the Gamma distribution. The precision of phase spectrum of the von Mises distribution is given by κ. If we set $\kappa = 0$, the Gamma von Mises Distribution degenerates into the Gamma Distribution. The pdf of x can be written as:

$$p(x_r, x_i) = \left|\tfrac{\partial(r,\theta)}{\partial(x_r,x_i)}\right|^{-1} p(r,\theta) = 2\beta e^{-\beta r} p(\theta) \tag{6.9}$$

where $x_0 = [r^{0.5\cos(\theta)}, r^{0.5\sin(\theta)}]$, $r = x_r^2 + x_i^2$, $\theta = \tanh^{-1}(\frac{x_i}{x_r})$. The decoder is

$$d_\theta(x|z) = 2\beta e^{-\beta r} = \frac{1}{2\pi\sigma^2} e^{-\frac{|x|^2}{2\sigma^2}} \tag{6.10}$$

We can write $\beta = \frac{1}{2\sigma^2}$. This decoder is a special case of the 2D-normal decoder where the mean $\mu = 0$. This gives us a direct connection between Type I and Type II decoders.

Itakura-Saito Divergence We derive the Itakura-Saito-VAE as the Gamma decoder. We assume the probabilistic decoder takes the form of a Gamma distribution.

$$d_\theta(x|z) = p(|x|) = \frac{1}{\Gamma(\alpha)} \frac{|x|^\alpha}{|y|^\alpha} |x|^{-2} e^{-\frac{|x|}{|y|}} \tag{6.11}$$

We take the negative log-likelihood of this distribution and we get our negative log-likelihood loss of,

$$-\log d_\theta(x|z) = \frac{|x|}{|y|} - \alpha \log \frac{|x|}{|y|} + \log \Gamma(\alpha) + 2 \log |x|. \tag{6.12}$$

In this formulation, x is the ground truth signal, y is the estimated signal, and α is a parameter of the distribution estimated by the NN.

6.5 Experiments

In this section we evaluate the effectiveness of different decoder distributions to learn speech by answering the following questions: (1) can we learn phase information directly in the VAE process, and does explicitly penalizing for phase lead to a better power spectrum representation? (2) does modeling time dependence help model magnitude and phase of speech? (3) do decoders have different compressive capacities?

Dataset The VCTK dataset consists of 110 English speakers both male and female with various accents resampled from 48 KHz to 16 KHz. We randomly select 10% of the speakers for the training dataset and the rest are the test dataset. The time-domain speech signals are converted to power spectrograms using the short-time Fourier transform (STFT) with a periodic Hann analysis window of length 64 ms (1,024 samples) and a hopsize of 128 samples. For fully connected architectures, the training set is then subsampled over time, every

10 frames. This results in a training set of 1% of the full dataset. For convolution-based networks, we use a random window of 5ms and directly use 1% of the dataset. The frequency signal is then separated into log power and complex phase components before being input to the NN.

Baselines In this chapter we propose several decoder distributions derived above to the widely excepted baseline decoder enforced by the Itakura-Saito divergence as the reconstruction loss in the VLB. We train a VAE with two fully connected layers in the encoder and two fully connected layers for the decoder on 1% of the VCTK Corpus dataset and test on the rest of the dataset. Alternatively, we use a 1D Convolution over the time axis of the STFT as well as adding a GRU to model time dependencies in the latent space of the VAE. For all VAEs, we use a latent space of size 32 unless otherwise stated. We use the Perceptual Evaluation of Speech Quality (PESQ) score (RBH01) and the Fréchet Audio Distance (FAD) (KZR18) as metrics for comparing different decoders. In our experiments, we considered a few different optimizers to train the VAEs. SGD + M took too long to converge, and Adam-style optimizers often diverged due to parameter instability. We found PSGD + M to be the most stable optimizer which reliably converged to a good solution (Li18a; Li18b; Li18c; Li22).

Experimental Results Our experiments demonstrate that using ground truth or reconstructed phase via the Griffin-Lim Algorithm with Gamma or Log-Normal von Mises decoders significantly outperform the IS-VAE in terms of PESQ and FAD across various network architectures (see Table 6.2). The Gamma decoder, which does not explicitly encode for phase, achieves the highest PESQ and lowest FAD using an FCN and 1D Convolution architecture respectively. These two models capture the spectrum information at high and low frequencies better than other architectures, including an Itakura-Saito-VAE (see 6.2). We omit the Complex Gaussian decoder from Table 6.2 as empirically we saw that $\mu \to 0$ and it carries little information for speech reconstruction.

The precision of the decoder distribution in a VAE-based speech processing model can be utilized as a measure of the extent to which the magnitude and phase information has been captured by the model. Precision, defined as the inverse of the variance of a distribution, characterizes the tightness with which the data is concentrated around the mean.

If the precision of the decoder distribution is high, this suggests that the decoder is producing outputs that are tightly concentrated around the mean, indicating that the phase information has been effectively captured by the model. Conversely, a low precision indicates that the decoder outputs are spread out, which may indicate that the phase information has not been captured well or has been lost during the encoding-decoding process.

Figure 6.3 (a) illustrates that the noise power has higher precision due to its stationarity and therefore higher predictability compared to the speech power spectra. Additionally, in (b) it is shown that the phase was learned with higher precision at low frequencies during the speech frames. However, the learned phases are still too noisy for high-quality speech reconstruction. Using ground truth or consistent phase through the Griffin-Lim Algorithm results in improved speech resynthesis compared to using the phase learned by either Type I or Type II decoders.

Overall, the proposed Gamma decoder improves PESQ by 42% (near-perfect PESQ of 4.4) and reduces FAD by 86% compared to IS-VAE.

6.5.1 Decoder Compression Capabilities

We evaluate the ability of various decoders to preserve the details of the spectrogram when compressing the data using a latent space of size 16, which results in a compression ratio of 48. As shown in Figure 6.4, the Gamma decoder produces the spectrogram with the most detailed features. The Log Normal + von Mises and IS decoders capture the general spectral details but are less rich in the region around 2 kHz (red horizontal line). The Gamma + von Mises decoder fails to preserve the speech features. This comparison highlights the fundamental differences in the compression capabilities among decoder distributions.

Table 6.2: Train VAEs on 1% of the VCTK dataset with STFT 3/4th overlap to train fully connected, convolutional and recurrent architectures with different decoder distributions. Evaluate PESQ and FAD using ground truth (GT) or Griffin Lim (GL) phase information.

Decoder Loss	**PESQ** ($\uparrow$ Better)		**FAD** ($\downarrow$ Better)	
Phase Source	*GT*	*GL*	*GT*	*GL*
Gamma FC	**4.4**	**3.8**	0.6	1.8
Log Normal Von Mises FC	3.8	3.6	1.5	2.0
Gamma Von Mises FC	3.7	3.5	2.5	3.0
Gamma Conv1d	4.3	3.6	**0.4**	**0.8**
Gamma Conv1d + GRU	4.2	3.5	0.8	1.3
Gamma Conv1d + BiGRU	4.1	3.3	0.8	1.4
Itakura-Saito FC **(baseline)**	3.1	2.8	4.3	5.4

It is worth mentioning that Log Normal von Mises and Gamma von Mises decoders must encode both magnitude and phase information into the latent space of 16, whereas Gamma and IS decoders only need to encode the magnitude information.

6.6 Conclusion

In conclusion, this study has demonstrated that the use of a Gamma decoder, which models the magnitude spectrum, in VAEs significantly improves speech resynthesis when compared to the state-of-the-art IS-VAE. While our research has shown promising results in directly learning phase information within the VAE, further work is necessary to achieve speech quality comparable to using ground truth phase. Our experiments have also established that VAEs with decoders that explicitly estimate the parameters of a negative log loss function perform better than those using the IS-divergence. Additionally, we have evaluated the performance of both phase-agnostic and phase-aware decoders on various types of VAEs such as fully connected, convolution, and recurrent architectures. Among these, the Gamma decoder, which does not explicitly encode for phase, achieves the highest PESQ and lowest FAD using an FCN and 1D Convolution architecture respectively. Thus, while the use of a Gamma decoder can lead to improved performance in speech resynthesis, the challenge of directly learning phase information within the VAE remains a subject of ongoing research.

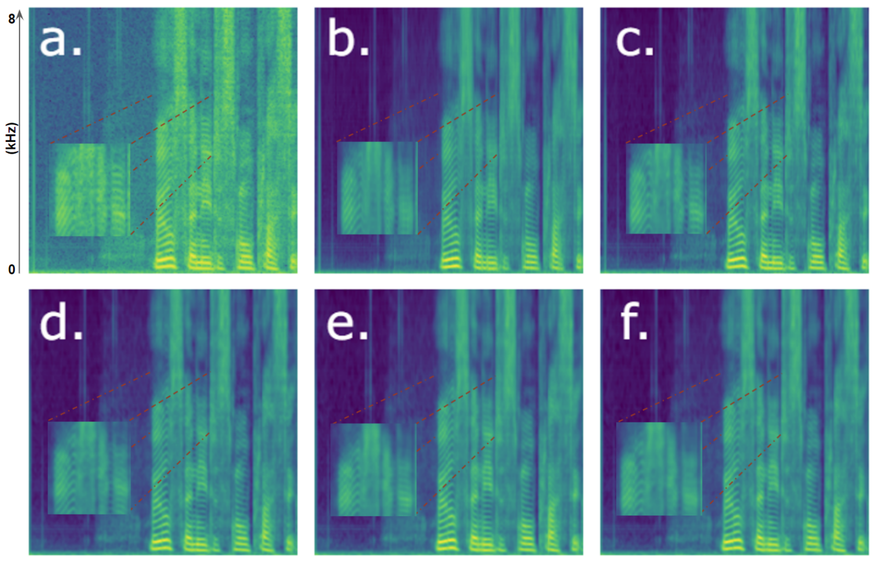
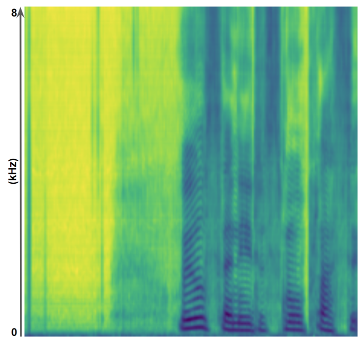
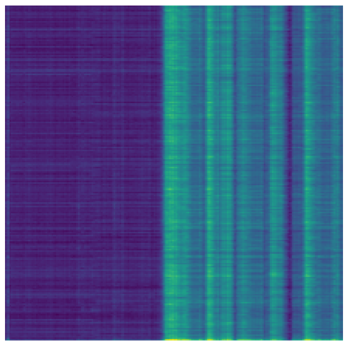


Figure 6.2: Effect of architecture on Gamma Decoder vs FCN IS-VAE. a. Ground Truth, b. IS-VAE, c. Gamma FC, d. Gamma Conv1d e. Conv1d GRU, f. Conv1d BiGRU. We see that the Gamma VAE FC and Conv1d capture high-fidelity speech patterns found in the ground truth signal as compared to other architectures and the IS-VAE.



(a) **Power Spectra Precision**



(b) **Phase Spectra Precision**

Figure 6.3: We see that the power spectrum of speech is being learned with high precision. We that the phase spectra are being learned in the frames of speech.

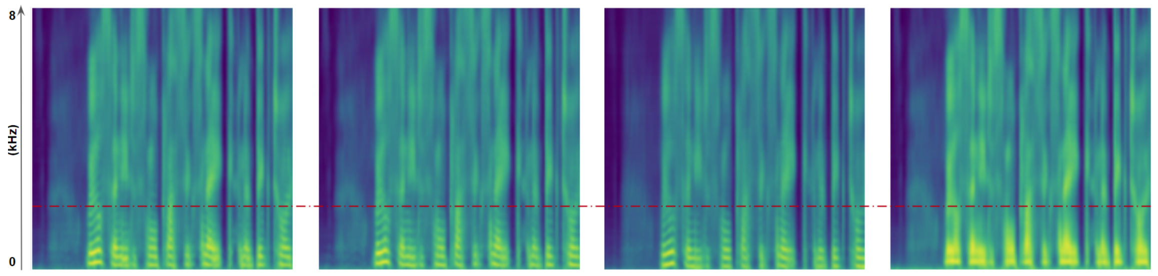


Figure 6.4: From left to right we have Gamma, Log Normal von Mises, Itakura-Saito, and Gamma von Mises decoder. We see that the Gamma distribution can capture the spectrogram at a high compression ratio with more detail. The Log Normal von Mises distribution captures speech features above the red dotted line whereas the IS decoder only captures features under it. With a high compression ratio, the Gamma von Mises fails to capture speech detail.

CHAPTER 7

Conclusions

7.1 Summary of Contributions

7.1.1 General Purpose Curvature-Informed Preconditioner

I believe the most impactful work of this dissertation is coming up with a novel, scalable, second-order preconditioner. In the context of this dissertation, we applied the preconditioner to SGD to improve the generalization capabilities of deep neural networks. This resulted in higher accuracies for classification networks and significantly lower perplexities for language models like LSTMs and GPT-2 style Transformers. We also showed that PSGD can preserve neuroplasticity in NNs far better than the SoTA optimization method. This will allow for better transfer learning as well as fine-tuning of models. Additionally, we saw that if we trained NNs with PSGD we no longer needed fundamental architectural advancements like the residual skip connection. I believe PSGD has the ability to become the go-to optimizer in ML.

In addition to standard optimization the preconditioner developed in this dissertation can be directly applied to many other places such as Langevin Dynamics for sampling from Energy-Bases and Diffusion models, Preconditioned Gradient Descent for Advaserial Attack, curvature-aware network pruning (see (WZG20)), etc.

7.1.2 Curvature-Informed Subset Selection

AdaCore is a curvature-aware subset selection method that builds on top of the Craig algorithm. NNs trained on AdaCore subsets outperform other SoTA subset selection methods in generalization while providing 4.5x speedup. This method can greatly reduce the computational overhead seen in ML training pipelines. We provide theoretical convergence guarantees as well as ample empirical evidence for our method.

7.1.3 Subset Selection for Self- & Semi-Supervised Generative Classifiers

We propose to leverage the generator of a LEBM as a learned means of data augmentation. That is while learning, conditional ancestral samples are drawn from the generative model to augment the labeled dataset. As the cardinality of such a set may be large, we propose to recruit coreset methods to reduce the set to a smaller coreset while maintaining an approximation of the cross-entropy gradient over the full augmented dataset. We show that an entropic regularization on the conditional samples is required to improve generalization. This is the first work to propose subset selection methods for self- or semi-supervised machine learning. In contrast to supervised learning where the unforgettable points are most important for generalization, we are the first to show that the forgettable points are important for model generalization for semi-supervised learning.

7.1.4 GammaVAE

In this dissertation, we explore the effect of decoder distribution on the re-synthesis ability of a VAE on speech data. We find that the Gamma distribution is exceptionally well-suited to modeling the distribution of speech. Using a NN for phase reconstruction is still an open task.

7.2 Future Work

7.2.1 Applications of General Preconditioner

The General purpose preconditioner we have developed in **Chaper 3** does not need to only be used for preconditioning SGD. In fact, it can be applied to any scenario to provide curvature information to the gradient.

Projected Gradient Descent Consider the standard Advaserial Attack (MMS17):

$$x^{t+1} = \Pi_{x+S}(x^t + \alpha sgn(\nabla_x L(\theta, x, y))) \tag{7.1}$$

can be easily converted to a preconditioned projected gradient descent attack via:

$$x^{t+1} = \Pi_{x+S}(x^t + \alpha sgn(P\nabla_x L(\theta, x, y))). \tag{7.2}$$

Riemann Manifold Metropolis Adjusted Langevin Algorithm Another situation where the general purpose preconditioner can be directly applied is Metropolis Adjusted Langevin Algorithm. Consider the random vector $\theta \in R^D$ with density $p(\theta)$ and denote the log density as $L(\theta) = \log \mathrm{p}(\theta)$, then the Metropolis Adjusted Langevin Algorithm (MALA) is based on a Langevin diffusion, with stationary distribution $\mathrm{p}(\theta)$, defined by the stochastic differential equation (SDE)

$$d\theta(t) = \frac{1}{2}\nabla_\theta L(\theta(t))dt + db(t) \tag{7.3}$$

where b denotes a D-dimension Brownian motion. A first-order Euler discretization of the SDE gives the following proposal mechanism:

$$\boldsymbol{\theta}^{n+1} = \boldsymbol{\theta}^{n} + \frac{\epsilon^2}{2} \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(\boldsymbol{\theta}^{n}\right) + \epsilon \mathbf{z}^{n} \tag{7.4}$$

where z $\mathcal{N}(\mathbf{z} \mid \mathbf{0}, \mathbf{I})$ and ϵ is the integration step size. This fundamental method has become the main way to do sampling in diffusion and energy-based models.

A theoretical counterpart to MALA is the Riemann Manifold Metropolis Adjusted Langevin Algorithm (RMMALA) which takes the local and global curvature information into account on the Riemannian manifold. In the most simple of applications, this method is reduced to a flat manifold with constant curvature becoming the pre-conditioned MALA proposal.

$$\boldsymbol{\theta}^{n+1} = \boldsymbol{\theta}^{n} + \frac{\epsilon^2}{2} \mathbf{P}\left(\boldsymbol{\theta}^{n}\right) \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(\boldsymbol{\theta}^{n}\right) + \epsilon \sqrt{\mathbf{P}\left(\boldsymbol{\theta}^{n}\right)} \mathbf{z}^{n} \tag{7.5}$$

This method can directly be implemented using the general purpose Lie Group preconditioner to improve the stability of the sampling scheme. Note, in RMMALA we are not only preconditioning the gradient but also the isotropic noise. Consider the following scenarios depicted in 7.5. We take a single image of a frog from the CIFAR10 dataset X and we repeat it 20 times. We run independent MCMC on the images using an energy-based model trained on the CIFAR10 dataset. We see that the preconditioned version of MCMC doesn't diverge where the standard version either severely degrades the image 7.5 (a) or totally diverges 7.5 (c).

Diffusion models utilize the same general equation with Langevin dynamics and exhibit the same instability (KAA22). Hence these results can be directly applied to improve the sampling ability of diffusion models, resulting in higher-quality sample generation.

BFGS Stye Rank updates to PSGD Convex optimization has many tools that can be directly used to improve PSGD. Right now we re-calculate the curvature information in PSGD with probability p, and in the meantime use a possibly outdated version of the curvature information to precondition the gradient. In BFGS low-rank updates are made to keep the curvature up to date between curvature re-calculations. I believe this will allow for faster convergence and potentially find better solutions.

Final Takeaways

- The Lie group is well suited to deal with noise.
- Use the preconditioner liberally – the LRA computation is fast and strong.
- For Deep CNNs (relatively flat) updating the preconditioner with p = 0.1 is sufficient.
- For Deep RNNs (highly curved) update the curvature as much as you can.
- For Transformers (sparse embedding) reduce lr preconditioner to accumulate information over time to beat sparsity

7.2.2 Causal Generative Classifier

At the core of machine learning models lies the objective of generalization, which involves the ability of the model to extend its learning beyond the training data. In the case of generative models, this entails the capacity to produce data that goes beyond the data used for training, with a desirable feature of controllability via disentanglement. We believe, that generative models can be considerably improved by integrating causal prior knowledge (i.e. via a causal graph). This integration will allow for principled generalization of generative models beyond the observed data, enabling out-of-distribution generalization that involves interventions and counterfactuals.

Although not discussed in depth in this dissertation, I have spent some time working on deep causal generative modeling. Similar to the rest of the dissertation, the generative model that was used to explore causal modeling was the VAE.

7.2.2.1 Causal Counterfactual Generative Models (CCGM)

Following the classic VAE model, given inputs $\mathbf{x}$, we encode into a latent space $\mathbf{z}$ with distribution q_ϕ where we have priors given by $p(\cdot)$.

$$\text{ELBO} = \mathbb{E}_{q_\mathcal{X}} \left[\mathbb{E}_{\mathbf{z} \sim q_\phi} \left[\log p_\theta(\mathbf{x} \mid \mathbf{z}) \right] - \mathcal{D} \left(q_\phi(\mathbf{z} \mid \mathbf{x}) \| p_\theta(\mathbf{z}) \right) \right] \tag{7.6}$$

In the Causal Counterfactual Generative Model (CCGM), the causal layer is described as a non-linear SCM

$$\mathbf{z}_i = g_i(\mathbf{A}_i \circ \mathbf{z}) \tag{7.7}$$

which finds some causal structure of the latent space variables $\mathbf{z}$ with respect to a matrix $\mathbf{A}$. Suppose $\mathbf{A}$ is composed of column vectors $\mathbf{A}_i$. For each latent space concept i, g_i is defined as a non-linear function $g_i : \mathbb{R}^n \to \mathbb{R}$. Where $\circ$ is the Hadamard product. In this formulation, $\mathbf{A}_i$ forms an $\mathbf{A}$ *adjacency* matrix. That is if $\mathbf{A}$ is viewed as a binary adjacency matrix, the g_i functions take the responsibility of reconstructing $\mathbf{z}$ given only the parents, dictated by $\mathbf{A}_i \circ \mathbf{z}$. The model balances a reconstruction loss, a latent loss that enforces an SCM, a standard VAE KL loss, as well as the differentiable DAG loss.

Our CCGM, allows for *counterfactual models and not just counterfactuals within the learned DAG*. By this, we mean one can modify a learned DAG by removing a connection and realizing an alternate DAG in which a "de-biasing" has occurred. The language here assumes that a learned connection is a "bias" within the true/learned generative model that we do not desire in our generated dataset - an implicit bias. In CCGM, the encoder directly generates the output $\mathbf{z}$, which is enforced to be standard-normally distributed. We pass this through our causal layer as one final *mutatable* bottleneck.

$$\mathbf{z} = \mathbf{A}^T \mathbf{z} \tag{7.8}$$

We solidify the structure of **A** by having exogenous and endogenous priors. This way, **A** can be split into a DAG term and a diagonal term

$$\mathbf{A} = \underbrace{\mathbf{G}}_{\text{DAG}} + \underbrace{\mathbf{B}}_{\text{diag.}} \tag{7.9}$$

where **B** has 1 on the diagonal for exogenous variables and 0 if endogenous. This ensures that the trivial solution where $\mathbf{A} = \mathbf{I}$ is never learned and enforces a causal relationship from the exogenous variables to the endogenous variables. So we assume parent/child priors are known.

Ultimately, this style of a causal latent-space model allows us to manipulate the latent space of a model to enforce priors and generalize to unseen data out of the training distribution. For more work not included in this dissertation on latent manipulation see (JPB22; BJP22; PJB23).

7.2.3 Next Steps for Causal Latent Manipulation:

I believe Latent Energy-Based Models will pair well with causal models. We have shown in this dissertation that LEBMs can be used in the self-supervised setting to learn augmentations that will boost classification accuracy as well as the generative quality of NNs. I believe a missing link that will allow us further boost classification and generation performance, obtain fine-grain control in sample generation, and allow for the debiasing of classifiers and generators is the integration of causality into our models.

Recall that the generative model resembles a variational auto-encoder with inference and generator model for which the latent prior is complexified by an energy-based model. The inference model allows for amortized inference of a latent vector given an observed example. The energy-based model couples the latent vector with a one-hot label vector, so that classification is based on the inferred latent vector for a given observed example. The generator model is required to learn the model by auto-encoding variational Bayes and is auxiliary as such.

Since the LEBM is basically a VAE whose latent prior has been complexified with an energy-based model we can directly add a causal layer in the latent space. This not only enables causal generation but also causal classification. Causal classification is novel and is important because it would allow us to directly de-bias and direct our classifiers.

We have some encoder $e(z|x)$ which transports x $\rightarrow$ z. We can then use the causal model from (JPB22; BJP22) to adjust z via $\hat{z} = g(A \circ z)$. Now that we have some causally manipulated version of z, using the LEBM, the soft-max classifier becomes

$$p_\alpha(y|\hat{z}) \propto \exp(\langle y, F_\alpha(\hat{z})\rangle) = \exp(F_\alpha(\hat{z})[y]). \tag{7.10}$$

To have a causal class conditional generation paired with latent Langevin dynamics one follows a similar procedure. We encode an image with label y via $e(z|x, y)$, and we then sample from the LEBM using the overdamped Langevin dynamics, which we run for T_{LD} steps with target distribution $p_\alpha(y, z)$,

$$z \sim p_0(z), \tag{7.11}$$

$$z_{t+1} = z_t + s\nabla_z \left[f_\alpha(z)[y] - \|z\|^2 \big/ 2\right] + \sqrt{2s}\epsilon_t,\ t = 1, \ldots, T_{LD} \tag{7.12}$$

Conclusion Generative modeling has taken the modern world of machine learning by storm. Introducing robust causal priors will allow us to generalize to out-of-distribution data as well as break potentially harmful biases in our data.

7.2.4 Closing Statements

Machine learning is a fast-moving and large research field today. It can be challenging to navigate all the new theoretical and empirical results as every day hundreds of new papers are uploaded to Arxiv. In my opinion, the best way to make progress is to read as many papers as you can on whatever topics interest you. If there is some seminal work in some subfield you don't necessarily find interesting, still read it. Eventually, the math and the intuition you build from reading these papers will allow you to draw new conclusions and make improvements to ML. Additionally, do not blindly believe everything you read just because it was published at a conference. There is a lot of conflicting research out there due to the rate of progress and experimental nature of the field. The best way to navigate it all is to consider your intuition, consider theory, and try out some experiments to see if you can get empirical results. At the end of the day, if you are learning, forming intuition, and having fun I believe the rest of the research will fall into place.

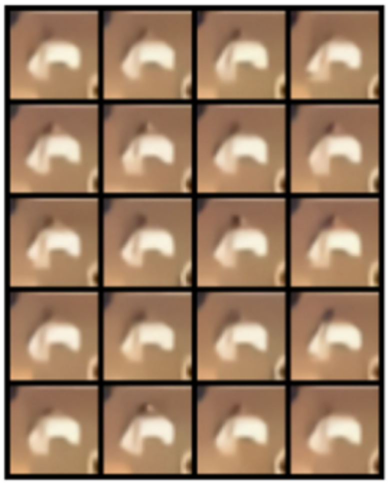


Figure 7.1: 1500 MCMC Steps

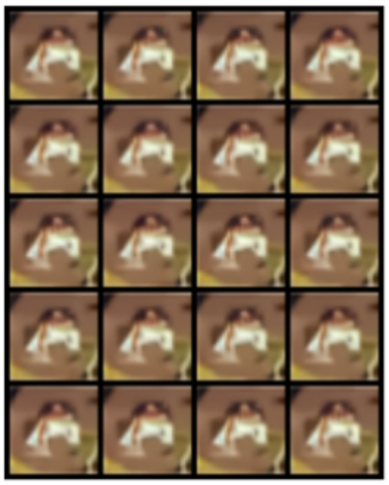


Figure 7.2: 1500 Precond MCMC Steps

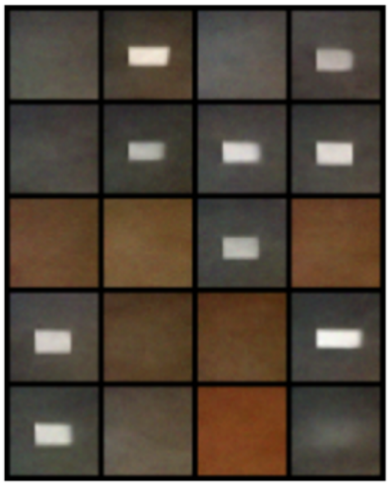


Figure 7.3: 15000 MCMC Steps

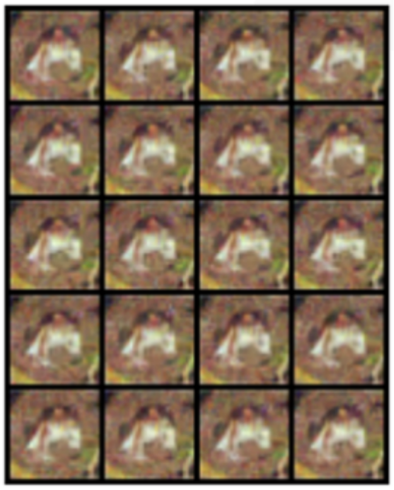


Figure 7.4: 15000 Precond MCMC Steps

Figure 7.5: Comparing standard MCMC sampling to Lie group curvature preconditioned MCMC sampling: We see that we sample from an energy-based model with $\epsilon = 1e - 2$ and MCMC temperature $t = 1e-5$ standard MCMC is unstable and results in severely degraded images which cannot be used. On the other hand, preconditioned MCMC is able to preserve stability.

APPENDIX A

Additional Proofs

A.1 Proofs for PSGD: Chapter 2

A.1.1 On the Convergence of PSGD

In this section, we show that the preconditioner P estimated by PSGD converges to the inverse of Hessian under mild conditions. We forgo convergence of network parameters since it is a well-studied topic in the literature.

To begin, let us formulate the problem clearly. Let (v, h) be a pair of vector and the associated noisy Hessian-vector product. We assume that v is drawn from distribution $\mathcal{N}(0, I)$. The noisy Hessian-vector product h is modeled by

$$h = H_0 v + \varepsilon$$

where H_0 is the true Hessian, and ε is a noise term independent of v due to use of sample averaged loss instead of the true loss. Then, we can write the criterion for the preconditioner estimation in PSGD as

$$\begin{aligned} c(P) =& E[h^T P h + v^T P^{-1} v] \\ =& \mathrm{tr}(P E[hh^T] + P^{-1} E[vv^T]) \\ =& \mathrm{tr}(P H^2 + P^{-1}) \end{aligned} \tag{A.1}$$

where

$$H^2 = H_0^2 + E[\varepsilon\varepsilon^T]$$

Clearly, H^2 is positive semi-definite regardless of the definiteness of H_0.

Note, we do not assume any definiteness or sparsity properties for H_0. Hence, in general, we assume that Q is fitted on a connected branch of the general linear group with learning rule

$$Q^{\text{new}} = Q^{\text{old}} + dQ = Q^{\text{old}} + Q\mathcal{E} \tag{A.2}$$

where Q relates to P as

$$P = Q^T Q,$$

and both dQ and $\mathcal{E}$ are sufficiently small matrices related by $dQ = Q\mathcal{E}$. One technical difficulty in proving the convergence of Q with the learning rule (A.2) is that it cannot exclude any rotation ambiguity of Q as suggested by

$$(UQ)^T(UQ) = Q^T(U^TU)Q = Q^TQ$$

where U can be any matrix on the orthogonal group. To ameliorate this technical issue, we constrain dQ to take on certain structure. In our proof, we assume that both Q^{old} and dQ are symmetric such that Q^{new} is always symmetric as well. To achieve such a constraint, we should update Q^{old} on the Lie group as

$$\begin{aligned} Q' =& Q^{\text{old}} + Q\mathcal{E} \\ Q^{\text{new}} =& [(Q')^T Q']^{0.5} \end{aligned} \tag{A.3}$$

In this way, $dQ = Q^{\text{new}} - Q^{\text{old}}$ is guaranteed to be symmetric as long as the starting guess Q^{old} is symmetric as well. Still, in practice we used the learning rule (A.2), and (A.3) only serves for the purpose of proof.

Assume that H is invertible, and $dQ = -\mu\frac{\partial c}{\partial Q}$ or $\mathcal{E} = -\mu Q^T \frac{\partial c}{\partial Q}$. Then, Q converges to $H^{-0.5}$ or $-H^{-0.5}$ with the learning rule (A.3) and a small enough positive step size μ.

Proof. Given a small perturbation dQ of Q, the change of P is given by

$$\begin{aligned} dP =&(Q + dQ)^T(Q + dQ) - Q^TQ \\ =&Q^TdQ + dQ^TQ + dQ^TdQ \end{aligned}$$

The change of P^{-1} is a little more complicated. We omit any terms smaller than $\mathcal{O}[(dQ)^2]$, and can expand dP^{-1} as below

$$\begin{aligned} dP^{-1} =&(P + dP)^{-1} - P^{-1} \\ =&-P^{-1}dPP^{-1} + P^{-1}dPP^{-1}dPP^{-1} \\ =&\left(-Q^{-1}dQP^{-1} - P^{-1}dQ^TQ^{-T} - P^{-1}dQ^TdQP^{-1}\right) \\ &+ (\ Q^{-1}dQQ^{-1}dQP^{-1} + Q^{-1}dQP^{-1}dQ^TQ^{-T} \\ &+ P^{-1}dQ^TdQP^{-1} + P^{-1}dQ^TQ^{-T}dQ^TQ^{-T}\ ) \\ =&-Q^{-1}dQP^{-1} - P^{-1}dQ^TQ^{-T} + Q^{-1}dQQ^{-1}dQP^{-1} \\ &+ Q^{-1}dQP^{-1}dQ^TQ^{-T} + P^{-1}dQ^TQ^{-T}dQ^TQ^{-T} \end{aligned}$$

Now, the change of the PSGD criterion is given by

$$\begin{aligned}
dc =& \mathrm{tr}(dPH^2 + dP^{-1}) \\
=& \mathrm{tr}(H^2 Q^T dQ + H^2 dQ^T Q + H^2 dQ^T dQ) \\
& + \mathrm{tr}(-Q^{-1} dQ P^{-1} - P^{-1} dQ^T Q^{-T} + Q^{-1} dQ Q^{-1} dQ P^{-1} \\
& + Q^{-1} dQ P^{-1} dQ^T Q^{-T} + P^{-1} dQ^T Q^{-T} dQ^T Q^{-T}) \\
=& 2\mathrm{tr}(H^2 Q^T dQ - P^{-1} Q^{-1} dQ) \\
& + \mathrm{tr}(dQ^T dQ H^2 + 2dQ Q^{-1} dQ P^{-1} Q^{-1} + dQ P^{-1} dQ^T Q^{-T} Q^{-1})
\end{aligned} \tag{A.4}$$

From (A.4), the first order derivatives of c with respect to Q is given by

$$\frac{\partial c}{\partial Q} = QH^2 - Q^{-T} P^{-1}$$

A stationary point is obtained by letting $\frac{\partial c}{\partial Q} = 0$, which leads to $H^2 = P^{-2}$. Hence, such a P always exists as long as H^2 is invertible. Via relationship $dQ = Q\mathcal{E}$, the gradient on the Lie group is

$$\nabla_{\mathcal{E}} = Q^T \frac{\partial c}{\partial Q}$$

Thus, fitting Q on the Lie group yields the same stationary point since Q is invertible. Note that the stationary points only tell us that $Q^T Q = H^{-1}$ without excluding the rotation ambiguity.

Without the constraint $dQ = dQ^T$, the second order derivative of c with respect to Q can be shown to be

$$\frac{\partial^2 c}{\partial(\mathrm{vec}(Q))^2} = 2I \otimes H^2 + 2J^T[Q^{-1} \otimes (QP)^{-T}] + 2[Q^{-T} \otimes (QP)^{-1}]J + 2(QQ^T)^{-1} \otimes P^{-1}$$

where J is the permutation matrix satisfying

$$\mathrm{vec}(dQ^T) = J\,\mathrm{vec}(dQ)$$

Via relationship $dQ = Q\mathcal{E}$, the second order derivative on the Lie group is

$$\nabla_{\mathcal{E}}^2 = (Q^T \otimes I)\frac{\partial^2 c}{\partial(\mathrm{vec}(Q))^2}(Q \otimes I) \tag{A.5}$$

We see that neither $\frac{\partial^2 c}{\partial(\mathrm{vec}(Q))^2}$ nor $\nabla_{\mathcal{E}}^2$ is guaranteed to be positive definite. Hence, the learning rule (A.2) does not convergence to a point as expected due to the rotation ambiguity.

On the other hand, both Q and dQ are symmetric with the learning rule (A.3). Thus, the second order derivative of c with respect to Q simplifies to

$$\frac{\partial^2 c}{\partial(\mathrm{vec}(Q))^2} = 2I \otimes H^2 + 2Q^{-1} \otimes (QP)^{-T} + 2Q^{-T} \otimes (QP)^{-1} + 2(QQ^T)^{-1} \otimes P^{-1}$$

At a stationary point, we have $Q = \pm P^{0.5} = \pm H^{-0.5}$. Thus, this second order derivative reduces to

$$\frac{\partial^2 c}{\partial(\mathrm{vec}(Q))^2}\,|_{Q=\pm H^{-0.5}} = 2I \otimes H^2 + 4H^{0.5} \otimes H^{1.5} + 2H \otimes H \tag{A.6}$$

By (A.5), the second order derivative on the Lie group is

$$\nabla_{\mathcal{E}}^2\,|_{Q=\pm H^{-0.5}} = 2H^{-1} \otimes H^2 + 4H^{-0.5} \otimes H^{1.5} + 2I \otimes H \tag{A.7}$$

Now, both $\frac{\partial^2 c}{\partial(\mathrm{vec}(Q))^2}$ and $\nabla_{\mathcal{E}}^2$ are positive definite for an invertible H at the stationary point. Thus, Q converges to either stationary point, i.e., $H^{-0.5}$ or $-H^{-0.5}$. $\square$

A.1.2 Construction of matrix-free preconditioner

The following statement gives a systematic way to construct a family of black-box matrix-free preconditioners.

Theorem 3.3.1. *Let $K = \{\sigma_1, \ldots, \sigma_m\}$ be a subgroup of the permutation group S_n. Then, linear transform $T : \mathbb{R}^n \mapsto \mathbb{R}^n$, $\quad T(x|a_1, \ldots, a_m) = \sum_{i=1}^{m} a_i \odot \sigma_i(x)$, forms a subgroup of $GL(n, \mathbb{R})$ parameterized with $\{a_1, \ldots, a_m\}$ if $T(\cdot|a_1, \ldots, a_m)$ is bijective, where both a_i and x are in $\mathbb{R}^n$.*

Proof. The proof follows by showing that such matrices have the following four properties required to form a Lie group.

First, we show that I is the identity element. Note that K has element e since it is a subgroup of S_n. Then without the loss of generality, we can set $\sigma_1 = e$ and a_1 to be a vector of ones, and all the other a_i, $i > 1$, to be vectors of zeros. This leads to $T(x|a_1, \ldots, a_m) = x$, and thus $T(\cdot|a_1, \ldots, a_m) = I$ when represented as a matrix.

Second, we show that such transforms are closed with binary operation $T^{(1)} \circ T^{(2)}$ defined as $[T^{(1)} \circ T^{(2)}](x) = T^{(1)}[T^{(2)}(x)]$. Specifically, we have

$$\begin{aligned}
[T^{(1)} \circ T^{(2)}](x) =& \sum_{i=1}^{m} a_i^{(1)} \odot \sigma_i^{(1)} \left(\sum_{j=1}^{m} a_j^{(2)} \odot \sigma_j^{(2)}(x) \right) \\
=& \sum_{i=1}^{m} \sum_{j=1}^{m} [a_i^{(1)} \odot \sigma_i^{(1)}(a_j^{(2)})] \odot \sigma_i^{(1)}(\sigma_j^{(2)}(x))
\end{aligned}$$

Since K is a subgroup of S_n, $\sigma_i^{(1)}(\sigma_j^{2}(\cdot))$ still belongs to K. Hence, $T^{(1)} \circ T^{(2)}$ will have the same form as $T(\cdot|a_1, \ldots, a_m)$ after merging like terms.

By representing $T(\cdot|a_1, \ldots, a_m)$ as a matrix, it is clear that the associativity property, i.e., $(T^{(1)} \circ T^{(2)}) \circ T^{(3)} = T^{(1)} \circ (T^{(2)} \circ T^{(3)})$, holds since matrix multiplication is associative. Lastly, the inverse of $T(\cdot|a_1, \ldots, a_m)$ exists since we assume that $T(\cdot|a_1, \ldots, a_m)$ is bijective, and thus its representation is an invertible matrix. □

Corollary A.1.1. *We want to point out that not all simple matrix-free preconditions can be constructed by Theorem 3.3.1. Let us take the widely used feature normalization transform, e.g., batch normalization (IS15), layer normalization (BKH16) and group normalization (WH18) as an example. We have*

$$T(x|\mu, \sigma) = (x - \mu)/\sigma$$

where μ and σ are either scalar or vector mean and standard deviation of x, respectively. This $T(\cdot|\mu, \sigma)$ form a sparse affine group for $\sigma \neq 0$ (Li19). However, we cannot use such preconditioners as black-box ones.

A.1.3 Construction of low-rank approximation preconditioners

The following property states that the proposed low-rank approximation preconditioners can fit both ends of the spectra of Hessian. It is not difficult to prove this statement. But, it reveals an important advantage over traditional low-rank approximation preconditions with form $P = \rho I + UU^T$, whose eigenvalues are lower bounded by ρ.

Notations

Let $Q = (I + UV^T)B$, where U and V are two tall thin matrices, and B is a matrix on certain group, e.g., the group of diagonal matrix, or simply a scalar. It is an important preconditioner as after reparameterization, we can have $Q = \text{diag}(d) + UV^T$ for diagonal matrix B, which relates to the LM-BFGS, HF, and conjugate gradient (CG) methods. This preconditioner is efficient if low-rank modication can significantly further reduce the condition number [1] after

[1]Since PH is not symmetric, smaller eigenvalue spread does not always suggest smaller condition number, e.g., $[1, a; 0, 1]$ has arbitrarily large conditioner number for $|a| \to \infty$. In PSGD, P does not amplify the gradient noise (ref (Li15), page 5-6, section IV.B), and thus avoids such degraded solution.

preconditioning the Hessian with a diagonal matrix, i.e., a Jacobi preconditioner. One also can think Q as a low-rank approximation of the inverse square root of a positive definite Hessian. Note that this form of preconditioner can fit both tails of the spectra of Hessian.

Theorem 3.3.2. *Preconditioner $P = Q^TQ$ with $Q = \rho(I + UV^T)$ can have positive eigenvalues arbitrarily larger than ρ^2 and arbitrarily smaller than ρ^2 with proper U and V.*

Proof. Let us check the simplest case, i.e., $\rho = 1$, $U = u$ and $V = v$. Then, P is shown to be

$$\begin{aligned} P =&(I + vu^T)(1 + uv^T) \\ =&I + vu^T + uv^T + (u^Tu)vv^T \end{aligned}$$

This P has two eigenvalues determined by u and v, say λ_1 and λ_2. They satisfy

$$\begin{aligned} \lambda_1\lambda_2 =&(1 + u^Tv)^2 \\ \lambda_1 + \lambda_2 =&2 + 2u^Tv + \|u\|^2\|v\|^2 \end{aligned}$$

By choosing u and v properly, these two eigenvalues can be arbitrarily smaller or larger than 1. For example, by letting $u^Tv = 0$ and $\|u\| = \|v\| \to \infty$, we have $\lambda_1\lambda_2 = 1$ and $\lambda_1+\lambda_2 \to \infty$. Hence, we must have one eigenvalue arbitrarily large, and the other one arbitrarily small. In general, the order of rank can be larger than 1, and thus more degree of freedoms for fitting the spectra of Hessian. □

Theorem 3.3.3. *If $\rho \neq 0$ and $(I + V^TU)^{-1}$ or $(I + U^TV)^{-1}$ exists, $A_V(\rho, U) = \rho(I + UV^T)$ defines a subgroup of $GL(n, \mathbb{R})$ parameterized with ρ and U. Similarly, $A_U(\rho, V) = \rho(I + UV^T)$ defines another subgroup of $GL(n, \mathbb{R})$ parameterized with ρ and V.*

Proof. Without the loss of generality, we assume $\rho = 1$, and simplify rewrite $A_V(1, U)$ as $A_V(U) = I + UV^T$. We can show that $A_V(U)$ forms a Lie group by revealing the following facts

$$\begin{aligned} A_V(0) =& I \\ A_V(U_1)A_V(U_2) =& A_V(U_1 + U_2 + U_1V^TU_2) \\ A_V^{-1}(U) =& A_V[-U(I + V^TU)^{-1}] \end{aligned}$$

i.e., the existence of identity element, closed with respect to matrix multiplication, and the existence of inverse, respectively. The last equation is simply the rewriting of the Woodbury matrix identity, i.e.,

$$[I + UV^T]^{-1} = I - U(I + V^TU)^{-1}V^T$$

The condition that $(I + V^TU)^{-1}$ exists is necessary as otherwise $A_V(U)$ is singular. Lastly, the associativity property clearly holds for matrix multiplications. Hence, all such matrices form a Lie group. Similarly, we can show that $A_U(V) = I + UV^T$ defines another Lie group.

□

The rotation ambiguity and Schur decomposition

Note that $UV^T = UQ(VQ)^T$ for any orthogonal matrix Q. Thus, we can remove this rotation ambiguity by selecting a Q such that $Q^T(V^TU)Q$ is an upper triangular block matrix with block size 1 or 2, i.e., the Schur decomposition. $A_V(U)$ and $A_U(V)$ still form groups by constraining V^TU to be quasi-triangular.

A.1.4 Low-rank approximation preconditioner fitting

In practice, we seldom use $Q = \rho(I + UV^T)$ as a preconditioner directly. Its limitation is clear, i.e., most of its eigenvalues are ρ^2 with $r \ll n$. In our method, we start from a rough preconditioner guess, say B, and modify it with $I + UV^T$ to have the refined preconditioner as

$$Q = (I + UV^T)B$$

If matrix B is from another Lie group, we still can update Q efficiently. For example, B can be a diagonal matrix with nonzero diagonals. Then this composited preconditioner reduce to a diagonal one when $r = 0$.

A.1.4.1 Fundamental operations on Lie Group

The concept of group generator

Unlike the additive update to move a point in a Euclidean space, we use multiplicative update to move a point on the Lie group. For example, we can move $A_V(U)$ to any its neighbor, say $A_V(U + \mathcal{E})$, as

$$A_V\left(\mathcal{E}(I + V^TU)^{-1}\right)A_V(U) = A_V(U + \mathcal{E})$$

Since $A_V(\mu U) = I + \mu UV^T = e^{\mu UV^T}$ for $\mu \to 0$, UV^T is a group generator for any $U \neq 0$. Indeed, the Lie algebra is closed as shown by

$$< U_1V^T, U_2V^T >= U_1V^TU_2V^T - U_2V^TU_1V^T = (U_1V^TU_2 - U_2V^TU_1)V^T$$

where $< \cdot, \cdot >$ is the Lie bracket.

The gradients for preconditioner updating

Here, the gradient always refers to the one on the Lie group. Thus dA, say on group $A_V(U)$, is either $A_V(\mathcal{E})A_V(U)$ or $A_V(U)A_V(\mathcal{E})$, where $\mathcal{E} \to 0$. Since we will update both groups, we simple put it as $dA = \mathcal{E}_1A$ or $dA = A\mathcal{E}_2$. Let's drop the E in the PSGD preconditioner fitting criterion and derive the stochastic gradient as below,

$$
\begin{aligned}
&d(h^T P h + v^T P^{-1} v)\\
=&h^T dP h - v^T P^{-1} dP P^{-1} v\\
=&2h^T dQ^T Q h - 2v^T P^{-1} dQ^T Q P^{-1} v
\end{aligned}
$$

Since we are to fit A and B on the Lie groups, thus let

$$
\begin{aligned}
dQ =& dAB + AdB\\
=&\mathcal{E}_1 AB + AB\mathcal{E}_2\\
=&\mathcal{E}_1 Q + Q\mathcal{E}_2
\end{aligned}
$$

Then, we have

$$
\begin{aligned}
&d(h^T P h + v^T P^{-1} v)\\
=&2h^T dQ^T Q h - 2v^T P^{-1} dQ^T Q P^{-1} v\\
=&2h^T Q^T \mathcal{E}_1^T Q h + 2h^T \mathcal{E}_2^T Q^T Q h - 2v^T P^{-1} Q^T \mathcal{E}_1^T Q P^{-1} v - 2v^T P^{-1} \mathcal{E}_2^T Q^T Q P^{-1} v\\
=&2h^T Q^T \mathcal{E}_1^T Q h + 2h^T \mathcal{E}_2^T P h - 2v^T Q^{-1} \mathcal{E}_1^T Q^{-T} v - 2v^T P^{-1} \mathcal{E}_2^T v\\
=&2\text{tr}\left\{\mathcal{E}_1^T \left[(Qh)(Qh)^T - (Q^{-T}v)(Q^{-T}v)^T\right]\right\} + 2\text{tr}\left\{\mathcal{E}_2^T \left[(Ph)h^T - v(P^{-1}v)^T\right]\right\} \qquad \text{(A.8)}\\
&\qquad \text{(A.9)}
\end{aligned}
$$

Gradient with respect to B

For the diagonal matrix, we simply have

$$
0.5\nabla_B = \text{diag}[(Ph)h^T - v(P^{-1}v)^T]
$$

from (A.8), and thus update B as

$$B \leftarrow B(I - \mu \nabla_B)$$

where the step size μ is small enough such that $\mu\|\nabla_B\| < 1$, and $\|\nabla_B\|$ is just the max absolute diagonal element for a diagonal matrix. Here, $\|\cdot\|$ denotes spectral norm.

For a diagonal matrix, we also can update its diagonals with element-wise step sizes as the Lie group reduces to the direct sum of a series smaller ones with dimension one. This could lead to faster convergence when the true Hessian is diagonal. Otherwise, we do not observe any significant difference between these two step size selection strategies in our numerical results.

Gradient with respect to U on group $A_V(U)$

Since

$$dA = (I + \mathcal{E}V^T)(I + UV^T) - (I + UV^T) = \mathcal{E}V^T A$$

we replace the $\mathcal{E}_1$ in (A.8) with $\mathcal{E}V^T$ to obtain gradient

$$0.5\nabla_U = [(Qh)(Qh)^T - (Q^{-T}v)(Q^{-T}v)^T]V$$

Then, we update A as

$$A \leftarrow A - \mu \nabla_U V^T A$$

which suggests its parameter can be updated as

$$U \leftarrow U - \mu \nabla_U (I + V^T U)$$

since we are operating on the group $A_V(U)$, where the step size is small enough such that $\mu\|\nabla_U V^T\| < 1$. Note that $\nabla_U V^T$ has at most rank two. Hence, its Frobenius norm can be used to bound its spectral norm tightly, as shown by

$$\|\nabla_U V^T\|_F/\sqrt{2} \le \|\nabla_U V^T\| \le \|\nabla_U V^T\|_F$$

Gradient with respect to V on group $A_U(V)$

As

$$dA = (I + U\mathcal{E}^T)(I + UV^T) - (I + UV^T) = U\mathcal{E}^T A$$

we replace the $\mathcal{E}_1$ in (A.8) with $U\mathcal{E}^T$ to give gradient

$$0.5\nabla_V = [(Qh)(Qh)^T - (Q^{-T}v)(Q^{-T}v)^T]U$$

Thus, we update A as

$$A \leftarrow A - \mu U \nabla_V^T A$$

which implies that its parameter V should be updated as

$$V \leftarrow V - \mu(I + VU^T)\nabla_V$$

where the step size is small enough such that $\mu\|U\nabla_V^T\| < 1$. Again, $U\nabla_V^T$ has at most rank two, and its Frobenius norm gives a tight enough estimation of its spectral norm.

A.2 Proofs for AdaCore: Chapter 3

A.2.1 Proof of Theorem 4.4.1

Theorem 4.4.1. *Assume that $\mathcal{L}$ is α-strongly convex and β-smooth. Let S be a weighted subset obtained by* AdaCore *that estimates the preconditioned gradient by an error of at most ϵ at every iteration t, i.e., $\|\mathbf{H}_t^{-1}\boldsymbol{g}_t - \sum_{j\in S}\gamma_{t,j}\mathbf{H}_{t,j}^{-1}\boldsymbol{g}_{t,j}\| \le \epsilon$. Then with learning rate*

α/β, Newton's method with update rule of Eq. (2.8) *applied to the subsets has the following convergence behavior:*

$$\mathcal{L}(w_{t+1}) - \mathcal{L}(w_t) \leq -\frac{\alpha^3}{2\beta^4}(\|\mathbf{g}_t\| - \beta\epsilon)^2. \tag{4.13}$$

In particular, the algorithm converges to a $\beta\epsilon/\alpha$-neighborhood of the optimal solution w_*.

Proof. We prove Theorem 4.4.1 (similarly to the proof of Newton's method in (BV04b)) for the following general update rule for $0 \leq k \leq 1$:

$$\Delta w_t = \mathbf{H}_t^{-k}\mathbf{g}_t \tag{A.10}$$

$$w_{t+1} = w_t - \eta\Delta w_t \tag{A.11}$$

For $k = 1$, this corresponds to the update rule of the Newton's method. Define $\lambda(w_t) = (\mathbf{g}_t^T\mathbf{H}_t^{-k}\mathbf{g}_t)^{1/2}$. Since $\mathcal{L}(w)$ is β-smooth, we have

$$\mathcal{L}(w_{t+1}) \leq \mathcal{L}(w_t) - \eta\mathbf{g}_t^T\Delta w_t + \frac{\eta^2\beta\|\Delta w_t\|^2}{2} \tag{A.12}$$

$$\leq \mathcal{L}(w_t) - \eta\lambda(w_t)^2 + \frac{\beta}{2\alpha^k}\eta^2\lambda(w_t)^2, \tag{A.13}$$

where in the last equality, we used

$$\lambda(w_t) = \Delta w_t\mathbf{H}_t^k\Delta w_t^T. \tag{A.14}$$

Therefore, using step size $\hat{\eta} = \frac{\alpha^k}{\beta}$ we have $w_{t+1} = w_t - \hat{\eta}\Delta w_t$

$$\mathcal{L}(w_{t+1}) \leq \mathcal{L}(w_t) - \frac{1}{2}\hat{\eta}\lambda(w_t)^2 \tag{A.15}$$

Since $\alpha I \preceq \mathbf{H}_t \preceq \beta I$, we have

$$\lambda(w_t)^2 = \mathbf{g}_t^T \mathbf{H}_t^{-k} \mathbf{g}_t \geq \frac{1}{\beta^k} \|\mathbf{g}_t\|^2, \tag{A.16}$$

and therefore $\mathcal{L}$ decreases as follows,

$$\mathcal{L}(w_{t+1}) - \mathcal{L}(w_t) \leq -\frac{1}{2\beta^k} \hat{\eta} \|\mathbf{g}_t\|^2 = -\frac{\alpha^k}{2\beta^{k+1}} \|\mathbf{g}_t\|^2. \tag{A.17}$$

Now for the subset, from Eq. (4.4) we have that $\|\mathbf{H}_t^{-1}\mathbf{g}_t - \sum_{j\in S} \gamma_{t,j} \mathbf{H}_{t,j}^{-1} \mathbf{g}_{t,j}\| \leq \epsilon$. Hence, via reverse triangle inequality $\|\mathbf{H}_t^{-1}\mathbf{g}_t\| \leq \|\sum_{j\in S} \gamma_{t,j} \mathbf{H}_{t,j}^{-1} \mathbf{g}_{t,j}\| + \epsilon$, and we get

$$\frac{\|\mathbf{g}_t\|}{\beta} \leq \|(\mathbf{H}_t)^{-1}\mathbf{g}_t\| \leq \|(\mathbf{H}_t^S)^{-1}\mathbf{g}_t^S \boldsymbol{\gamma}\| + \epsilon \leq \frac{\|\mathbf{g}_t^S\|}{\alpha} + \epsilon, \tag{A.18}$$

where $\mathbf{g}_t^S = \sum_{j\in S} \mathbf{g}_{t,j}$ and $\mathbf{H}_t^S = \sum_{j\in S} \mathbf{H}_{t,j.}$ are the gradient and Hessian of the subset respectively. In Eq. (A.18) the RHS follows from operator norms and the LHS follows from the following lower bound on the norm of the product of two matrices:

$$\begin{aligned} \|AB\| &= \max_{\|x\|=1} \|x^T AB\| \\ &= \max_{\|x\|=1} \|x^T A\| \left\| \frac{x^T A}{\|x^T A\|} B \right\| \\ &\geq \max_{\|x\|=1} \sigma_{\min(A)} \left\| \frac{x^T A}{\|x^T A\|} B \right\| \\ &= \max_{\|y\|=1} \sigma_{\min(A)} \left\| y^T B \right\| \\ &= \sigma_{\min(A)} \|B\|, \end{aligned} \tag{A.19}$$

Hence,

$$\|\mathbf{g}_t^S\| \geq \frac{\alpha}{\beta}(\|\mathbf{g}_t\| - \beta\epsilon) \tag{A.20}$$

Therefore, on the subset we have

$$\mathcal{L}(w_{t+1}) - \mathcal{L}(w_t) \leq -\frac{\alpha^k}{2\beta^{k+1}}\|\mathbf{g}_t^S\|^2 \tag{A.21}$$

$$\leq -\frac{\alpha^k}{2\beta^{k+1}}(\frac{\alpha}{\beta})^2(\|\mathbf{g}_t\| - \beta\epsilon)^2 \tag{A.22}$$

$$= -\frac{\alpha^{k+2}}{2\beta^{k+3}}(\|\mathbf{g}_t\| - \beta\epsilon)^2. \tag{A.23}$$

The algorithm stops descending when $\|\mathbf{g}_t\| = \beta\epsilon$. From strong convexity we know that

$$\|\mathbf{g}_t\| = \beta\epsilon \geq \alpha\|w - w_*\| \tag{A.24}$$

Hence, we get

$$\|w - w_*\| \leq \beta\epsilon/\alpha. \tag{A.25}$$

As such we have Corollary 4.4.2. and when we set $k = 1$ we have proof of Theorem 4.4.1. □

Descent property for Equation 4.4 For a strongly convex function, $\mathcal{L}$, we have that the diagonal elements of the Hessian are positive (YGS20). This allows the diagonal to approximate the full Hessian which has good convergence properties.

Given a function $\mathcal{L}(w)$ which is strongly convex and strictly smooth, we have that $\nabla^2\mathcal{L}(w)$ is upper and lower bounded by two constants β and α, so that $\alpha I \leq \nabla^2\mathcal{L}(w) \leq \beta I$, for all w. For a strongly convex function the diagonal elements in diag($\mathbf{H}_t$) are all positive, and we have:

$$\alpha \leq e_i^T\mathbf{H}_t e_i = e_i^T\text{diag}(\mathbf{H}_t)e_i = \text{diag}(\mathbf{H}_t)_{i,j} \leq \beta \tag{A.26}$$

where e_j represents the natural basis vectors. Therefore, the diagonal entries of diag($\mathbf{H}_t$)

are in the range $[\alpha, \beta]$. Therefore, the average of a subset of the numbers is still in the range $[\alpha, \beta]$. As such, we can prove that Eq. (4.9) has the same convergence rate as its full matrix counterpart, by following the same proof as Theorem 4.4.1.

A.2.2 Proof of Theorem 4.4.3 and 4.4.4

A loss function $\mathcal{L}(w)$ is considered μ-PL on a set $\mathcal{S}$, if the following holds:

$$\frac{1}{2}\|\mathbf{g}\|^2 \geq \mu\left(\mathcal{L}(w) - \mathcal{L}\left(w_*\right)\right), \forall w \in \mathcal{S} \tag{A.27}$$

where w_* is a global minimizer. When additionally $\mathcal{L}\left(w_*\right) = 0$, the μ-PL condition is equivalent to the μ-PL* condition

$$\frac{1}{2}\|\mathbf{g}\|^2 \geq \mu\mathcal{L}(w), \forall w \in \mathcal{S}. \tag{A.28}$$

Theorem 4.4.3. *Assume that the loss function $\mathcal{L}(w)$ is β-smooth, and μ-PL* on a set $\mathcal{W}$, and S is a weighted subset obtained by* AdaCore *that estimates the preconditioned gradient by an error of at most ϵ, i.e., $\|\mathbf{H}_t^{-1}\boldsymbol{g}_t - \sum_{j\in S}\gamma_{t,j}\mathbf{H}_{t,j}^{-1}\boldsymbol{g}_{t,j}\| \leq \epsilon$. Then with learning rate η, gradient descent with update rule of Eq. (4.2) applied to the subsets have the following convergence behavior at iteration t:*

$$\mathcal{L}(w_t) \leq (1 - \frac{\eta\mu\alpha^2}{\beta^2})^t\mathcal{L}(w_0) - \frac{\eta\alpha^2}{2\beta^2}(\beta^2\epsilon^2 - 2\beta\epsilon\nabla_{\max}), \tag{4.15}$$

where α is the minimum eigenvalue of all Hessian matrices during training, and $\nabla_{\max}$ is an upper bound on the norm of the gradients.

For Lipschitz continuous $\mathbf{g}$ and μ-PL* condition, gradient descent on the entire dataset yields

$$\mathcal{L}(w_{t+1}) - \mathcal{L}(w_t) \leq -\frac{\eta}{2}\|\mathbf{g}_t\|^2 \leq -\eta\mu\mathcal{L}(w_t), \tag{A.29}$$

and,

$$\mathcal{L}(w_t) \leq (1-\eta\mu)^t \mathcal{L}(w_0) \tag{A.30}$$

which was shown in (LZB20).

We build upon this result to an AdaCore subset.

Proof. From Eq. (A.18) we have that

$$\frac{\|\mathbf{g}_t\|}{\beta} \leq \|(\mathbf{H}_t)^{-1}\mathbf{g}_t\| \leq \|(\mathbf{H}_t^S)^{-1}\mathbf{g}_t^S\boldsymbol{\gamma}\| + \epsilon \leq \frac{\|\mathbf{g}_t^S\|}{\alpha} + \epsilon \tag{A.31}$$

Hence solving for $\|\mathbf{g}_t^S\|$ we have,

$$\|\mathbf{g}_t^S\| \geq \frac{\alpha}{\beta}(\|\mathbf{g}_t\| - \beta\epsilon). \tag{A.32}$$

For the subset we have

$$\mathcal{L}(w_{t+1}) - \mathcal{L}(w_t) \leq -\frac{\eta}{2}\|\mathbf{g}_t^S\|^2 \tag{A.33}$$

By substituting Eq. (A.32) we have.

$$\leq -\frac{\eta\alpha^2}{2\beta^2}(\|\mathbf{g}_t\| - \beta\epsilon)^2 \tag{A.34}$$

$$= -\frac{\eta\alpha^2}{2\beta^2}(\|\mathbf{g}_t\|^2 + \beta^2\epsilon^2 - 2\beta\epsilon\|\mathbf{g}_t\|) \tag{A.35}$$

$$\leq -\frac{\eta\alpha^2}{2\beta^2}(\|\mathbf{g}_t\|^2 + \beta^2\epsilon^2 - 2\beta\epsilon\nabla_{\max}) \tag{A.36}$$

$$\leq -\frac{\eta\alpha^2}{2\beta^2}(2\mu\mathcal{L}(w_t) + \beta^2\epsilon^2 - 2\beta\epsilon\nabla_{\max}) \tag{A.37}$$

Where we can upper bound the norm of $\mathbf{g}_t$ in Eq. (A.35) by a constant ∇_{max}. And Eq. (A.37) follows from the μ-PL condition from Eq. (A.27).

Hence,

$$\mathcal{L}(w_{t+1}) \leq (1 - \frac{\eta\mu\alpha^2}{\beta^2})\mathcal{L}(w_t) - \frac{\eta\alpha^2}{2\beta^2}(\beta^2\epsilon^2 - 2\beta\epsilon\nabla_{\max}) \tag{A.38}$$

Since, $\sum_{j=0}^{k}(1 - \frac{\eta\mu\alpha^2}{\beta^2})^j \leq \frac{\beta^2}{\eta\mu\alpha^2}$, for a constant learning rate η we get

$$\mathcal{L}(w_{t+1}) \leq (1 - \frac{\eta\mu\alpha^2}{\beta^2})^{t+1}\mathcal{L}(w_0) - \frac{\eta\alpha^2}{2\beta^2}(\beta^2\epsilon^2 - 2\beta\epsilon\nabla_{\max}) \tag{A.39}$$

□

Theorem 4.4.4. *Under the same assumptions as in Theorem 4.4.3, for mini-batch SGD with mini-batch size $m \in \mathbb{N}$, the mini-batch SGD with update rule Eq. (4.2), with learning rate $\eta = \frac{m}{\beta(m-1)}$, applied to the subsets have the following convergence behavior:*

$$\mathbb{E}[\mathcal{L}(w_t)] \leq (1 - \frac{\eta\mu\alpha^2}{2\beta})^t\mathbb{E}[\mathcal{L}(w_0)] - \frac{\alpha^2\eta}{2\beta}(\beta\epsilon^2 - 2\epsilon\nabla_{\max}) \tag{4.16}$$

where α is the minimum eigenvalue of all Hessian matrices during training, and $\nabla_{\max}$ is an upper bound on the norm of the gradients, and the expectation is taken w.r.t. the randomness in the choice of mini-batch.

For Lipschitz continuous $\mathbf{g}$ and μ-PL* condition, gradient descent on the entire dataset yields

$$\mathcal{L}(w_{t+1}) - \mathcal{L}(w_t) \leq -\frac{\eta}{2}\|\mathbf{g}_t\|^2 \leq -\eta\mu\mathcal{L}(w_t), \tag{A.40}$$

and,

$$\mathcal{L}(w_t) \leq (1-\eta\mu)^t\mathcal{L}(w_0), \tag{A.41}$$

which was shown in (LZB20).

We build upon this result to an AdaCore subset.

Proof. From Eq. (A.18) we have that

$$\frac{\|\mathbf{g}_t\|}{\beta} \leq \|(\mathbf{H}_t)^{-1}\mathbf{g}_t\| \leq \|(\mathbf{H}_t^S)^{-1}\mathbf{g}_t^S\boldsymbol{\gamma}\| + \epsilon \leq \frac{\|\mathbf{g}_t^S\|}{\alpha} + \epsilon \tag{A.42}$$

Hence solving for $\|\mathbf{g}_t^S\|$ we have,

$$\|\mathbf{g}_t^S\| \geq \frac{\alpha}{\beta}(\|\mathbf{g}_t\| - \beta\epsilon). \tag{A.43}$$

For the subset we have

$$\mathcal{L}(w_{t+1}) - \mathcal{L}(w_t) \leq -\frac{\eta}{2}\|\mathbf{g}_t^S\|^2 \tag{A.44}$$

Fixing w_t and taking expectation with respect to the randomness in the choice of the batch $i_t^{(1)} \dots i_t^{(m)}$ (noting that those indices are i.i.d.), we have

$$\mathbb{E}_{i_t^{(1)} \dots i_t^{(m)}}[\mathcal{L}(w_{t+1}) - \mathcal{L}(w_t)] \leq -\eta(\alpha - \eta\frac{\beta}{m}(\alpha\frac{m-1}{2} + \beta)\mathcal{L}(w_t) \tag{A.45}$$

$$\leq -\underbrace{\eta(1 - \frac{\eta\beta(m-1)}{2m})}_{c_1}\|\mathbf{g}_t\|^2 + \underbrace{\frac{\eta^2\beta\lambda}{m}}_{c_2}\mathcal{L}(w_t) \tag{A.46}$$

$$\leq -c_1\frac{\alpha^2}{\beta^2}\|\mathbf{g}_t + \beta\epsilon\|)^2 + c_2\mathcal{L}(w_t) \tag{A.47}$$

We can upper bound the norm of $\mathbf{g}_t$ in Eq. (A.37) by a constant ∇_{max}. And Eq. (A.37) follows from the μ-PL condition from Eq. (A.27) and assuming $\eta \leq \frac{2}{\beta}$.

$$\leq -c_1\frac{\alpha^2}{\beta^2}(\mu\mathcal{L}(w_t) - 2\nabla_{max}\beta\epsilon + \beta^2\epsilon^2) + c_2\mathcal{L}(w_t) \tag{A.48}$$

$$\leq -\eta(1 - \frac{\eta\beta(m-1)}{2m})\frac{\alpha^2}{\beta^2}(\mu\mathcal{L}(w_t) - 2\nabla_{max}\beta\epsilon + \beta^2\epsilon^2) + \frac{\eta^2\beta\lambda}{m}\mathcal{L}(w_t) \tag{A.49}$$

$$= \eta(\mu\frac{\alpha^2}{\beta^2} - \eta\frac{\beta}{m}(\mu\frac{\alpha^2(m-1)}{\beta^2 2} + \lambda))\mathcal{L}(w_t) + \frac{\eta^2\beta\lambda}{m}\mathcal{L}(w_t) + c_1\frac{\alpha^2}{\beta^2}(2\nabla_{max}\beta\epsilon - \beta^2\epsilon^2) \tag{A.50}$$

$$= \eta\mu\frac{\alpha^2}{\beta^2}(1 - \eta\beta\frac{m-1}{2m})\mathbb{E}[\mathcal{L}(w_t)] + \eta\frac{\alpha^2}{\beta^2}(1 - \eta\beta\frac{m-1}{2m})(2\nabla_{max}\beta\epsilon - \beta^2\epsilon^2) \tag{A.51}$$

By optimizing the quadratic term in the upper bound with respect to η we get $\eta = \frac{m}{\beta(m-1)}$.

$$\mathbb{E}[\mathcal{L}(w_{t+1})] \leq (1 - \frac{\mu\alpha^2 m}{2\beta^2(m-1)})\mathbb{E}[\mathcal{L}(w_t)] + \frac{\alpha^2 m}{\beta^2}\frac{2\nabla_{max}\beta\epsilon - \beta^2\epsilon^2}{2\beta(m-1)} \tag{A.52}$$

Hence,

$$\mathbb{E}[\mathcal{L}(w_{t+1})] \leq \left(1 - \frac{\eta^*(m)\mu\alpha^2}{2\beta}\right)\mathbb{E}[\mathcal{L}(w_t)] + \frac{\alpha^2\eta^*(m)}{\beta}(\nabla_{max}\epsilon - \beta\epsilon^2/2) \tag{A.53}$$

$\square$

A.2.3 Discussion on Greedy to Extract Near-optimal Coresets

As discussed in Section 4.4.4, a greedy algorithm can be applied to find near-optimal coresets that estimate the general descent direction with an error of at most ϵ by solving the submodular cover problem Eq. (4.12). For completeness, we include the pseudocode of the greedy algorithm in Algorithm 3. The AdaCore algorithm is run per class.

Algorithm 3 AdaCore (Adaptive Coresets for Accelerating First and Second Order Optimization Methods)

```
Require: Set of component functions $f_i$ for $i \in V = [n]$.
Ensure:  Subset $S \subseteq V$ with corresponding per-element stepsizes $\gamma j \in S$.
$S_0 \leftarrow \emptyset, s_0 = 0, i = 0.$;
while $F(S) < C_1 - \epsilon$ do
    $j \in \arg\max_{e \in V \setminus S_{i-1}} F(e|S_{i-1})$;
    $S_i = S_{i-1} \cup j$;
    $i = i + 1$;
end while
for $j = 1$ to $|S|$ do
    $\gamma_j = \sum_{i \in V} \mathbb{I}\big[j = \arg\min_{s \in S} \max_{w \in \mathcal{W}} |\mathbf{H}_t^{-1}\mathbf{g}_t - \sum_{j \in S} \gamma t, j \mathbf{H}_{t,j}^{-1}\mathbf{g}_{t,j}|\big]$
end for
```

A.2.4 Bounding the Norm of Difference Between Preconditioned Gradients

A.2.4.1 Convex Loss Functions

We show the normed difference for ridge regression. Similar results can be deduced for other loss functions such as square loss (AYS16), logistic loss, smoothed hinge losses, etc.

For ridge regression $f_i(w) = \frac{1}{2}(\langle x_i, w \rangle - y_i)^2 + \frac{\lambda}{2}\|w\|^2$, we have $\nabla f_i(w) = x_i(\langle x_i, w \rangle - y_i) + \lambda w$. Furthermore for invertable Hessian H, let $H_i^{-1} = A$ and $H_j^{-1} = B$. Therefore,

$$\|A\nabla f_i(w) - B\nabla f_j(w)\| = \|A(x_i\langle x_i, w \rangle - x_i y_i + \lambda w) - B(x_j\langle x_j, w \rangle - x_j y_j + \lambda w)\| \quad \text{(A.54)}$$

$$= \|Ax_i\langle x_i, w\rangle - Bx_j\langle x_j, w\rangle + Bx_jy_j - Ax_iy_i + \lambda(A-B)w\| \tag{A.55}$$

$$\begin{aligned} &= \|Ax_i\langle x_i, w\rangle + Bx_j\langle x_i, w\rangle - Bx_j\langle x_i, w\rangle - Bx_j\langle x_j, w\rangle \\ &\qquad + Bx_jy_j - Ax_iy_i + Bx_jy_i - Bx_jy_i + \lambda(A+B)w\| \end{aligned} \tag{A.56}$$

$$= \|\langle x_i, w\rangle(Ax_i - Bx_j) + \langle x_i - x_j, w\rangle Bx_j + (y_j - y_i)Bx_j + y_i(Bx_j - Ax_i) + \lambda(A-B)w\| \tag{A.57}$$

$$= \|(\langle x_i, w\rangle - y_i)(Ax_i - Bx_j) + (\langle x_i - x_j, w\rangle + y_j - y_i)Bx_j + \lambda(A+B)w\| \tag{A.58}$$

$$\leq |\langle x_i, w\rangle - y_i|\|Ax_i - Bx_j\| + |\langle x_i - x_j, w\rangle + y_j - y_i|\|Bx_j\| + \lambda\|(A-B)w\| \tag{A.59}$$

$$\begin{aligned} &\leq |\langle x_i, w\rangle - y_i|(\|A-B\|\|x_i\| + \|B\|\|x_i - x_j\|) + |\langle x_i - x_j, w\rangle + y_j - y_i|\|Bx_j\| \\ &\qquad + \lambda\|(A+B)w\| \end{aligned} \tag{A.60}$$

$$\leq O(\|w\|)(\|A-B\| + \|B\|\|x_i - x_j\|) + O(\|w\|)\|B\|\|x_j\|\|x_i - x_j\| \tag{A.61}$$

$$\leq O(\|w\|)(\|A\| + \|B\| + \|B\|\|x_i - x_j\|) + O(\|w\|)\|B\|\|x_j\|\|x_i - x_j\| \tag{A.62}$$

In Eq. (A.62) we have the norm of the inverse of the Hessian matrix. Since H is invertible we have $\min_i \sigma_i > 0$,

$$\min_i \sigma_i = \inf_{x \neq 0} \frac{\|Hx\|_2}{\|x\|_2} \iff \frac{1}{\min_i \sigma_i} = \sup_{x \neq 0} \frac{\|x\|_2}{\|Hx\|_2} \tag{A.63}$$

$$\frac{1}{\min_i \sigma_i} = \sup_{x \neq 0} \frac{\|x\|_2}{\|Hx\|_2} = \sup_{H^{-1}z \neq 0} \frac{\|H^{-1}z\|_2}{\|z\|_2} = \sup_{z \neq 0} \frac{\|H^{-1}z\|_2}{\|z\|_2} = \left\|H^{-1}\right\|_2, \tag{A.64}$$

where the substitution $Hx = z$ was made, and utilized that $H^{-1}z = 0 \iff z = 0$ since H is invertible. Hence,

$$\leq O(\|w\|)\|B\|\|x_i - x_j\| \tag{A.65}$$

$$\leq O(\|w\|)\|x_i - x_j\| \tag{A.66}$$

For $\|x_i\| \leq 1$, and $|y_i - y_j| \approx 0$.

Assuming that $\|w\|$ is bounded for all $w \in \mathcal{W}$, an upper bound on the euclidean distance between preconditioned gradients can be precomputed.

A.2.4.2 Neural Networks

We closely follow proofs from (KF18) and (MBL20) to show that we can bound the difference between the Hessian inverse preconditioned gradients of an entire NN up to a constant of the difference between the Hessian inverse preconditioned gradients of the last layer of the NN, between arbitrary datapoints i and j.

Consider an L-layer perceptron, where $w^{(l)} \in \mathbb{R}^{M_l x M_{l-1}}$ is the weight matrix for the l^{th} layer with M_l hidden units. Furthermore assume $\sigma^{(l)}(.)$ is a Lipschitz continuous activation function. Then we let,

$$x_i^{(0)} = x_i, \tag{A.67}$$

$$z_i^{(l)} = w^{(l)} x_i^{(l-1)}, \tag{A.68}$$

$$x_i^{(l)} = \sigma^{(l)}\left(z_i^{(l)}\right). \tag{A.69}$$

With,

$$\Sigma_l'\left(z_i^{(l)}\right) = \operatorname{diag}\left(\sigma'^{(l)}\left(z_{i,1}^{(l)}\right), \cdots \sigma'^{(l)}\left(z_{i,M_l}^{(l)}\right)\right) \tag{A.70}$$

$$\Delta_i^{(l)} = \Sigma_l'\left(z_i^{(l)}\right) w_{l+1}^T \cdots \Sigma_l'\left(z_i^{(L-1)}\right) w_L^T. \tag{A.71}$$

We have,

$$\|\mathbf{H}_i^{-1}\mathbf{g}_i - \mathbf{H}_j^{-1}\mathbf{g}_j\| \tag{A.72}$$

$$= \left\| \left(\Delta_i^{(l)} \Sigma_L'(z_i^{(L)})(\mathbf{H}_i^{-1})^{(L)}\mathbf{g}_i^{(L)}\right)(x_i^{(l-1)})^T - \left(\Delta_j^{(l)} \Sigma_L'(z_j^{(L)})(\mathbf{H}_j^{-1})^{(L)}\mathbf{g}_j^{(L)}\right)(x_j^{(l-1)})^T \right\| \tag{A.73}$$

$$\leq \left\|\Delta_i^{(l)}\right\| \cdot \left\|x_i^{(l-1)}\right\| \cdot \left\|\Sigma_L'\left(z_i^{(L)}\right)(\mathbf{H}_i^{-1})^{(L)}\mathbf{g}_i^{(L)} - \Sigma_L'\left(z_j^{(L)}\right)(\mathbf{H}_j^{-1})^{(L)}\mathbf{g}_j^{(L)}\right\| \tag{A.74}$$
$$+ \left\|\Sigma_L'\left(z_j^{(L)}\right)(\mathbf{H}_i^{-1})^{(L)}\mathbf{g}_i^{(L)}\right\| \cdot \left\|\Delta_i^{(l)}\left(x_i^{(l-1)}\right)^T - \Delta_j^{(l)}\left(x_j^{(l-1)}\right)^T\right\|$$

$$\leq \left\|\Delta_i^{(l)}\right\| \cdot \left\|x_i^{(l-1)}\right\| \cdot \left\|\Sigma_L'\left(z_i^{(L)}\right)(\mathbf{H}_i^{-1})^{(L)}\mathbf{g}_i^{(L)} - \Sigma_L'\left(z_j^{(L)}\right)(\mathbf{H}_j^{-1})^{(L)}\mathbf{g}_j^{(L)}\right\| \tag{A.75}$$
$$+ \left\|\Sigma_L'\left(z_j^{(L)}\right)(\mathbf{H}_i^{-1})^{(L)}\mathbf{g}_i^{(L)}\right\| \cdot \left(\left\|\Delta_i^{(l)}\right\| \cdot \left\|x_i^{(l-1)}\right\| + \left\|\Delta_j^{(l)}\right\| \cdot \left\|x_j^{(l-1)}\right\|\right)$$

$$\leq \underbrace{\max_l \left(\left\|\Delta_i^{(l)}\right\| \cdot \left\|x_i^{(l-1)}\right\|\right)}_{c_{l,i}} \cdot \left\|\Sigma_L'\left(z_i^{(L)}\right)(\mathbf{H}_i^{-1})^{(L)}\mathbf{g}_i^{(L)} - \Sigma_L'\left(z_j^{(L)}\right)(\mathbf{H}_j^{-1})^{(L)}\mathbf{g}_j^{(L)}\right\| \tag{A.76}$$
$$+ \underbrace{\left\|\Sigma_L'\left(z_i^{(L)}\right)(\mathbf{H}_i^{-1})^{(L)}\mathbf{g}_i^{(L)}\right\| \cdot \max_{l,i,j}\left(\left\|\Delta_i^{(l)}\right\| \cdot \left\|x_i^{(l-1)}\right\| + \left\|\Delta_j^{(l)}\right\| \cdot \left\|x_j^{(l-1)}\right\|\right)}_{c_2}$$

From (KF18), (MBL20), we have that the variation of the gradient norm is mostly captured by the gradient of the loss function with respect to the pre-activation outputs of the last layer of our neural network. Here we have a similar result, where, the variation of the gradient preconditioned on the inverse of the Hessian norm is mostly captured by the gradient preconditioned on the inverse of the Hessian of the loss function with respect to the pre-activation outputs of the last layer of our neural network. Assuming $\left\|\Sigma_L'\left(z_i^{(L)}\right)(\mathbf{H}_i^{-1})^{(L)}\mathbf{g}_i^{(L)}\right\|$ is bounded, we get

$$\|\mathbf{H}_i^{-1}\mathbf{g}_i - \mathbf{H}_j^{-1}\mathbf{g}_j\| \leq c_1 \left\|\Sigma_L'\left(z_i^{(L)}\right)(\mathbf{H}_i^{-1})^{(L)}\mathbf{g}_i^{(L)} - \Sigma_L'\left(z_j^{(L)}\right)(\mathbf{H}_j^{-1})^{(L)}\mathbf{g}_j^{(L)}\right\| + c_2 \tag{A.77}$$

where c_1, c_2 are constants. The above holds for an affine operation followed by a slope-bounded non-linearity ($|\sigma'(w)| \leq K$).

A.2.5 Analytic Hessian for Logistic Regression

Here we provide the analytical formulation of the Hessian of the binary cross entropy loss per data point n with respect to weights $\mathbf{w}$ for Logistic Regression.

For Binary Logistic Regression we have a loss function $\mathcal{L}(\mathbf{w})$ defined as:

$$\mathcal{L}(\mathbf{w}) = -\sum_{i=1}^{N} l_i(\mathbf{w}) = -\sum_{i=1}^{N} y_i ln(\hat{\sigma}) + (1-y_i)ln(1-\hat{\sigma}), \text{ where } \hat{\sigma}_i = \sigma(\mathbf{w^T x_i} + b) \quad \text{(A.78)}$$

and σ is the sigmoid function.

We form a Hessian matrix for each data point i based on loss function $\ell_i(\mathbf{w})$ as follows:

$$H_n = \left(\begin{array}{c|c} \frac{\partial}{\partial \mathbf{w^2}} l_i(\mathbf{w}) & \frac{\partial}{\partial \mathbf{w} \partial b} l_i(\mathbf{w}) \\ \hline \frac{\partial}{\partial b \partial \mathbf{w}} l_i(\mathbf{w}) & \frac{\partial}{\partial b \partial b} l_i(\mathbf{w}) \end{array}\right) = \left(\begin{array}{c|c} \hat{\sigma}_i(1-\hat{\sigma}_i)\mathbf{x_i x_i^T} & \hat{\sigma}_i(1-\hat{\sigma}_i)\mathbf{x_i} \\ \hline [\hat{\sigma}_i(1-\hat{\sigma}_i)\mathbf{x_i}]^T & \hat{\sigma}_i(1-\hat{\sigma}_i) \end{array}\right)$$

This allows us to analytically form the Hessian information per point which is needed to precompute a single coreset which will be used throughout training of the convex regularized logistic regression problem.

APPENDIX B

Additional Experiments

B.1 More Experimental Results: Chapter 3

B.1.1 MNIST Handwriting Digit Recognition

To have a fair comparison between the diagonal and low-rank approximation (LRA) preconditioners, we slightly upgrade Q in the LRA preconditioner to form

$$Q = \mathrm{diag}(d)(I + UV^T)$$

where d is a vector. This form of Q cannot form a Lie group. Still, its two factors, $\mathrm{diag}(d)$ and $I + UV^T$, can be fitted on their own Lie groups. Now, the diagonal preconditioner is a special case of this LRA one with order $r = 0$.

We have tested orders $r = 0, 1, 2, 5$, and 10. The batch size is 64. Totally ten epochs of training are performed. The learning rate for parameter updating is annealed exponentially from 0.1 for the first epoch to 0.001 for the tenth epoch. The step size for preconditioner fitting is annealed exponentially from 0.1 for the first epoch to 0.01 for the tenth epoch. Preconditioned gradient norm is clipped to 10 if too large. No momentum is used. Figure 1 summarizes the test classification error rates for preconditioners with different order of approximations over 50 runs. From Fig. 1, we see that the simple diagonal preconditioner performs well. Still, LRA brings marginal gains up to order $r = 5$. This cost function is fairly ‘flat’ since the only nonlinearities in LeNet5 are two piece-wise linear functions, i.e., the activation function ReLU and max pooling one. Only the cross entropy loss introduces the 'curvature'.

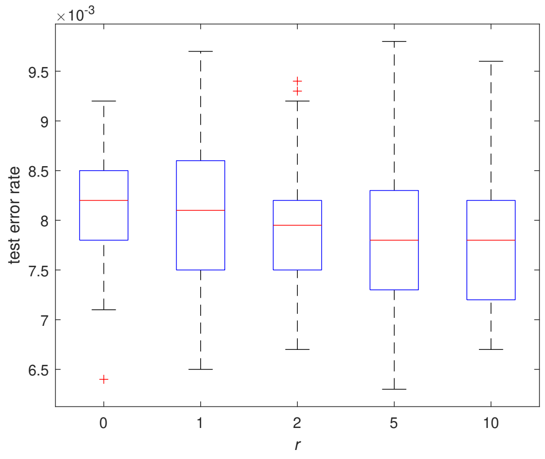


Figure B.1: MNIST test classification error rates over 50 runs using preconditioners with different orders of LRA. The one with order 0 reduces to the diagonal preconditioner. Table 1 reports the results of classification accuracy for $r = 5$. Higher is better.

B.1.2 CIFAR10 Image Classification with ResNet18

We follow the implementations of Adabelief[1] algorithm (ZTD20) to test preconditioned SGD (PSGD) on the CIFAR10 image classification task with the ResNet18 model. One main difference from the implementations in (ZTD20) is that we reduce the learning rate by tenfold twice for all the optimizers, while the original stage learning rate scheduler only anneals the step size once. We also consider the cosine learning rate scheduler, which helps SGD to achieve the state-of-the-art (SOTA) test accuracy of about 95.5%. Training and testing accuracy convergence curves over 16 runs are plotted in Fig. 2. We only show the results of PSGD and SGD here as SGD is known to achieve the SOTA results for this problem.

For PSGD, we use step size 0.02 for parameter updating and 0.01 for preconditioner fitting. The preconditioner is only updated once per ten iterations, and thus its overhead

[1]https://github.com/juntang-zhuang/Adabelief-Optimizer

over SGD is marginal. The same momentum factor, 0.9, is used for both SGD and PSGD. Since the step size in PSGD is normalized, we update the momentum as $m \leftarrow 0.9m + 0.1g$, instead of $m \leftarrow 0.9m + g$ as in the SGD. No gradient clipping is used. Weight decay is realized by adding the L2 regularization term $0.5\lambda\theta^T\theta$ to the cross entropy loss. We have found that λ between 0.01 and 0.02 performs the best.

From Fig. 2, we observe that SGD performs very well with the cosine learning rate scheduler. This is expected as these residual networks are highly evolved to be first-order optimizer-friendly. The extensive use of piece-wise linear functions, residual connections and batch normalizations make these models fairly 'flat' and resemble shallow models, instead of deep ones (VWB16). Still, PSGD slightly outperforms SGD when we remove the shortcut connections or use a less tuned learning rate scheduler, e.g., the stage one here.

B.1.2.1 PSGD is robust to hyper-parameters

To showcase the robustness of PSGD to hyper-parameters we show in table B.1 the test classification accuracy of a ResNet18 trained on CIFAR10 using different learning rate and weight decay.

PSGD XOR	Learning Rate	Weight Decay	Test Accuracy
XMat	2e-2	2e-2	95.32
LRA	2e-2	2e-2	95.69
LRA	5e-2	2e-2	95.48
LRA	5e-2	2e-2	95.46
LRA	4e-2	2e-2	95.55
LRA	3e-2	2e-2	95.57
LRA	2e-2	1e-2	95.45
LRA	2e-2	3e-2	95.51

Table B.1: We see that PSGD is robust to hyper-parameter selection making tuning significantly easier compared to other optimization methods.

We clearly see the test accuracy of PSGD in a wide range or learning rate and weight

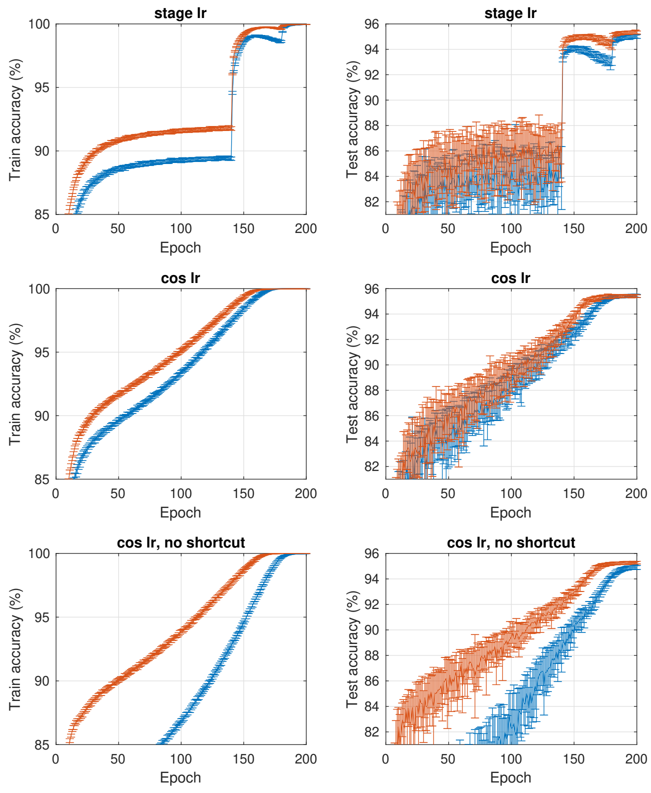


Figure B.2: CIFAR10 image classification with ResNet18. The order of low-rank Hessian approximation is 10. Mean and variance are estimated over 16 runs. Higher is better. decay converges within the expected rage of $95.49 \pm 0.08\%$.

B.1.3 A Large-Scale Logistic Regression Problem

We consider a simple logistic regression model to solve the MNIST classification problem, with and without Bernoulli noise. We considered AdaHessian, AdaBelief, SGD, LBFGS, PSGD LRA/XMat, KFAC, and HessianFree optimizers. Let x be the vector of the image with length 28^2. Instead of regression on vector x, we do the regression on the outer product vector of x, which has length 28^4. This significantly increases the test classification accuracy but leads to a large regression matrix with over six million coefficients. The KFAC preconditioner did not fit on our GPU and the HessianFree optimizer would diverge far from all other optimizers in more than 50% of our runs. Both are omitted from further discussion in this section.

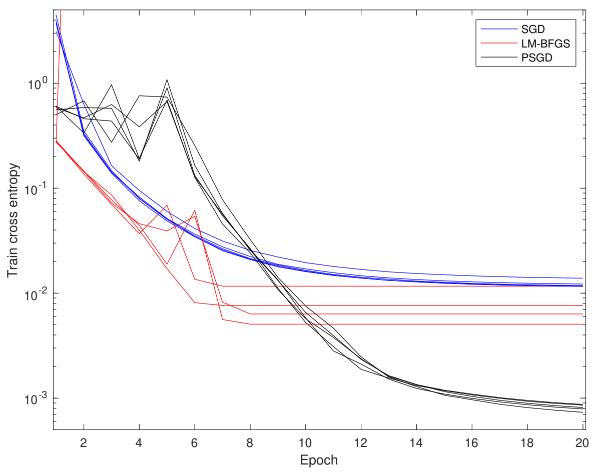
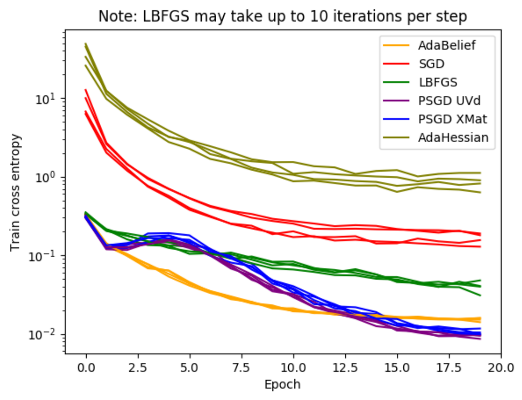


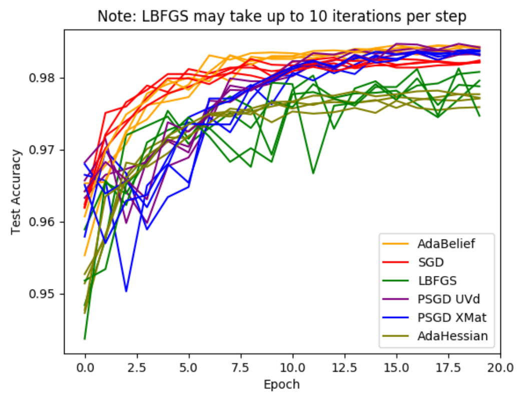
Figure B.3: Standard MNIST dataset: Typical convergence curves on the logistic regression problem. Lower is better. When comparing the convergence speed, one should be aware that one step of LM-BFGS may take up to ten iterations, while SGD and PSGD always one iteration per step.

No momentum is considered since this is the case for LM-BFGS. The train batch size is 500. It is tricky to select the initial learning rate for LM-BFGS even though we exponentially anneal it. We have found that LM-BFGS diverges on roughly one-third of the trials with an initial learning rate of 0.1, but 0.05 is too small and may lead to worse performance than SGD. For PSGD, we consider the LRA preconditioner with order 10 and set the learning rates for parameters and preconditioner to 0.05 and 0.1, respectively. Since LM-BFGS might diverge with a learning rate of 0.1, we only show a few typical convergence curves of SGD, LM-BFGS, and PSGD in Fig. B.3. LM-BFGS converges to regression losses a few times smaller than SGD. PSGD could converge to losses about one order of magnitude lower than that of SGD and LM-BFGS. Regarding test classification error rate, we have $2.37\% \pm 0.08$, $2.09\% \pm 0.18$, $1.98\% \pm 0.08$ for SGD, LM-BFGS, and PSGD, respectively, averaged over ten runs. Again, LM-BFGS outperforms SGD, and PSGD performs the best on the test classification error rate as well.

Next, we simply randomly add Bernoulli noise to the MNIST dataset to add diversity to the dataset. Even though LM-BFGS is the standard optimizer for large-scale logistic regression we wanted to consider some other SOTA optimizers for deep learning. PSGD UVd/XMat performed the best in this scenario but we found that AdaBelief does surprisingly well given its lack of second-order information. Losses and Accuracies plotted in Fig B.4



(a) **Train Loss**



(b) **Test Accuracy**

Figure B.4: We consider logistic regression on the MNIST dataset with Bernoulli noise. We see PSGD finds the lowest loss on the train set, with the Belief mechanism also working well. PSGD and AdaBelief generalize well to the Accuracy of the Test Dataset.

We see that PSGD significantly outperforms the SOTA optimizers in the convex logistic regression setting under noise-free or noisy domains while being memory efficient.

B.1.4 Forgettability & Uncertainty: Rank Analysis

Very recently, (TSC18) shows that Forgettable points are crucial for generalization in the supervised setting. In the unsupervised, semi-supervised, self-supervised, and generative setting (PKN23) shows that one can use the low entropy samples generated by the generative model to significantly boost the classification performance of the Latent Energy Based Model which acts as generative-classifier. Here we acknowledge the previous findings and focus on the supervised setting. We train a ResNet18 on CIFAR-10 and record entropy and forgettability statistics. We plot the strong correlation between low forgettability score and low entropy found in our statistics in Fig. B.5 and provide correlation coefficients in Table B.2.

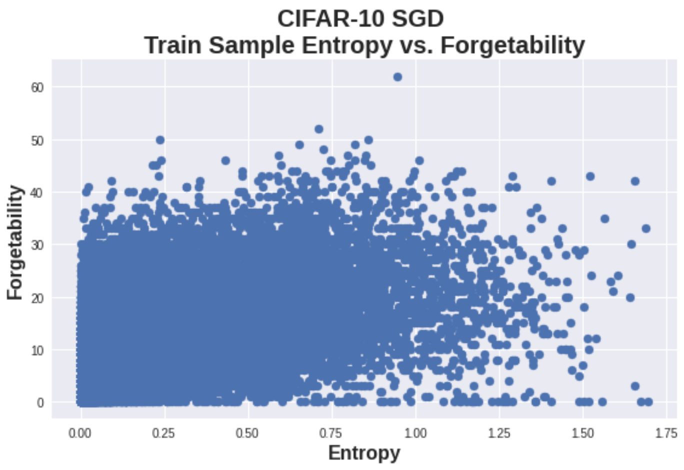


Figure B.5: There is a strong correlation between the forgettability ordering and the entropy ordering. i.e. points that are unforgettable have a very low entropy.

Entropy v Forgettability Score	Correlation Coefficient		p-value	
	SGD	PSGD	SGD	PSGD
Spearman	0.72	0.73	0	0
Pearson	0.57	0.65	0	0
Kedal Tau	0.54	0.56	0	0
Weighted Tau	0.60	0.75	0	0

Table B.2: Comparison of correlation metrics comparing Entropy Score vs Forgettability score between SGD and PSGD. We see that PSGD's average correlation between entropy and forgettability is stronger than that of SGD.

B.1.4.1 Generalization Gap Between High and Low Entropy Subsets

We train an oracle LeNet5 on the full MNIST dataset for 20 epochs, we categorize the train set into two subsets; a high entropy subset and a low entropy subset, each consisting of 10k points. The entropy is defined over the softmax of the logits. We then train two different LeNet5's on each dataset. We find that the low entropy dataset does not generalize to the test set well achieving a test accuracy of 74%, whereas the high entropy dataset achieves a 99.3% which is on par with oracle LeNet5 (see B.3). This clearly shows for supervised learning the high entropy points are most important for generalization.

Furthermore, we find that training on low entropy points results in low entropy a lower mean entropy network. This supports the hypothesis that there are certain points that the net can lower the entropy over which do not lead to generalization. In fact we see in Table B.3 that training over the full dataset resulted in a higher mean low entropy compared to that of the full dataset. This supports the idea that high entropy points are important for supervised learning.

In terms of distance from initialization, we see that the network is pushed farther from initialization when using the full dataset than while using the high entropy points and the least when using the low entropy points. This supports that training with low-entropy or unforgettable points unnecessarily pushes a network's parameters far from initialization reducing generalization capacity. Furthermore, we see that when training on low entropy points, PSGD does not push the neural network far from initialization preserving generalization capacity (CG19).

MNIST Statistics	Accuracy	Expected Low Entropy	Expected High Entropy	Distance from Initialization SGD	Distance from Initialization PSGD
Full Dataset	99.3%	5e-16	6e-3	23.1	26
High Entropy Subset	99.3%	1e-10	2e-2	21.7	21.2
Low Entropy Subset	74.2%	6e-18	4e-4	9.2	8.7

Table B.3: Comparing generalization accuracy of a LeNet5 trained on either 10k high or 10k low entropy datapoints. We see high entropy datasets can match the generalization of the full dataset with a higher test entropy compared to the other datasets. We see that the distance from initialization is less for the low entropy points.

Finally, we see that a NN trained on low entropy points has mean low entropy 2 orders of magnitude smaller than training on the full dataset, and 8 orders of magnitude smaller than training on the high entropy subset. This shows that low entropy points cause a level of confidence in a NN which is not needed and in some cases can be hurtful to generalization 3.5.1.

B.1.5 Effect of Rank on XOR

The full rank version of PSGD (Li19) can solve the XOR problem with an RNN past delay of 128. Since we can arbitrarily increase the rank of Hessian approximation in our LRA version of PSGD, we consider the effect of rank on convergence on the XOR problem using an RNN. We find rank approximations of $r = 0, 1, 2, 5$, and 10 converge with probability $p = 0.1, 0.4, 0.8, 1$ and 1 respectively.

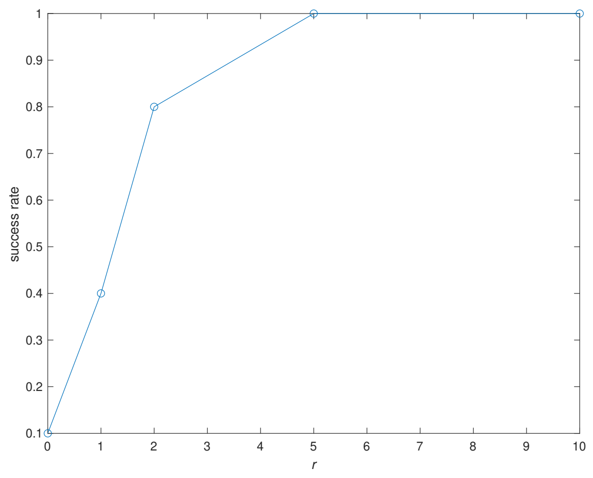


Figure B.6: Success rate over ten runs in solving the XOR problem with a simple RNN and LRA preconditioners of orders 0, 1, 2, 5, and 10. Higher is better.

This example shows a typical problem where the diagonal preconditioner struggles, while a low-order Hessian approximation works perfectly for preconditioning.

For all experiments, both step sizes for parameter and preconditioner updating are fixed at 0.01. The gradient clipping threshold is set to 1. No momentum is used. We run 10 runs per optimizer for $100,000$ iterations or until convergence. The success rates over ten runs for each r are plotted in Fig. B.6. This example shows a typical problem where the diagonal preconditioner struggles, while a low-order Hessian approximation works perfectly for preconditioning.

B.2 Further Empirical Evidence: Chapter 4

B.2.1 AdaCore estimates full gradient closely, reaching smaller loss

AdaCore obtains a better estimate of the preconditioned gradient by considering curvature and gradient information compared to the state-of-the-art algorithm Craig and random subsets. This is quantified by calculating the difference between the weighted gradients of coresets and the gradient of the complete dataset.

Fig B.7, shows the difference in loss reached by AdaCore vs Craig over different subset sizes. This shows that corsets selected using AdaCore to classify the Ijcnn1 dataset using logistic regression can reach a lower loss over varying subset sizes than Craig.

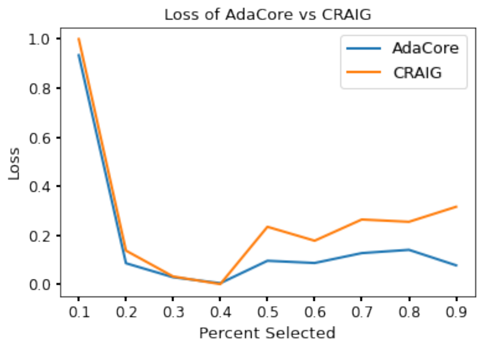


Figure B.7: Normalized loss for Logistic Regression over different subset sizes on Ijcnn1 dataset using SGD. AdaCore corsets, considering curvature information, to classify Ijcnn1 dataset using logistic regression consistently reaches a lower loss compared to CRAIG, which only considers the gradient information.

B.2.2 Class imbalance CIFAR-10

To provide further empirical evidence, we include results using a class-imbalanced version of the CIFAR-10 dataset for ResNet18. We skewed the class distribution linearly, keeping 90% of class 9, 80% of class 8 . . . 10% of class 1, and 0% of class 0, and trained for 200 epochs. Selecting a coreset for every epoch can be computationally expensive; instead, one can compute a coreset once every R epoch. Here we investigate AdaCore's performance on various R values. As Table B.4 shows, AdaCore can withstand class imbalance much better than Craig and randomly selected subsets. When $R = 20$, AdaCore achieves 57.3% final test accuracy, +8.7% above Craig, +2.6% above Random, 27.4% above GradMatch and 36.2% above Glister.

Table B.4: CIFAR-10 Class Imbalance, ResNet18. Final test accuracy and percent of full data selected (in parentheses). Trained with SGD + Momentum, selecting a coreset every R epochs that is S percent of the full dataset. Note AdaCore has greater accuracy while seeing fewer data points.

Accuracy	$S = 1\% R = 20$	$S = 1\% R = 10$	$S = 1\% R = 5$
AdaCore	**57.3%** ± 0.5 (**5%**)	**57.12** ± 0.96 (**9.5%**)	**60.2%** ± 0.36 (**14.5%**)
Craig	48.6% ± 0.8 (8%)	55 ± 1 (16%)	53.05% ± 0.24 (27.5%)
Random	54.7% ± 0.3 (8%)	54.6 ± 0.76 (18%)	54.6% ± 0.74 (33.2%)
GradMatch	29.9% ± 0.4 (8.2%)	29.1% ± 0.8 (14.7%)	32.75% ± 0.83 (23.2%)
Glister	21.1% ± 0.42 (8.6%)	17.2% ± 0.75 (16%)	14.4% ± 0.83 (22.2%)

Not only is AdaCore able to outperform Craig, random, GradMatch, and Glister, but it can do so while selecting a smaller fraction of the data points during training, as shown under all settings in Table B.4.

B.2.3 Class imbalance BDD100k

Additionally, we compared AdaCore to Craig and random selection for the BDD100k dataset, which has seven inherently imbalanced classes and 100k data points. We train ResNet50 with SGD + momentum for 100 epochs choosing subset size (s = 10%) every (R = 20) epoch on the weather prediction task. We see that AdaCore can outperform Craig by 2% and random by 8.8% seen in Table B.5.

Table B.5: AdaCore outperforms other baseline subset selection algorithms as well as training on the full dataset, reaching a better accuracy in less time. This provides up to a 2.3x speedup compared to the state of the art.

SGD + Momentum Accuracy	**S = 10% R = 20**
AdaCore	74.3%
Craig	72.3%
Random	65.5%

Additionally, Table B.6 shows that AdaCore outperforms baseline methods on BDD100k providing 2.3x speedup vs. training on the entire dataset and a 1.8x speedup vs. random. We see that Craig, GradMatch & Glister do not reach the accuracy of AdaCore even given more time and epochs. The epoch value is seen in parenthesis by accuracy. These experiments were run with SGD+momentum.

Table B.6: AdaCore outperforms other baseline subset selection algorithms as well as training on the full dataset, reaching a better accuracy in less time. This provides up to a 2.3x speedup compared to the state of the art.

	BDD100k		Speedup over	
$S = 10\%$ R = 20	Accuracy (epoch)	Time (s)	Rand	Full
AdaCore	74.3%(100)	7331	**1.8**	**2.3**
Craig	73.1%(150)	10996	1.3	1.6
Random	73.3%(180)	13050	1	1.2
GradMatch	72%(200)	14040	.7	1.1
Glister	73%(200)	12665	1.03	1.2
Full Dataset	74.3%(45)	16093	0.8	1

B.2.4 CIFAR-100

Table B.7 shows that AdaCore outperforms baseline methods on CIFAR100, providing 4x speedup vs. training on the entire dataset and a 3.8x speedup vs. Random. We see that Craig, GradMatch, and Glister do not reach the accuracy of AdaCore even given more time and epochs. The epoch value is seen in parenthesis by accuracy. These experiments were run with SGD+momentum.

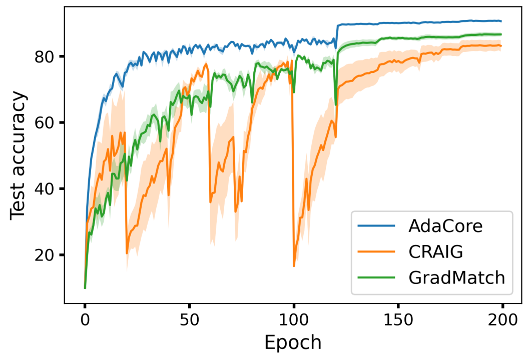
B.2.5 When first order coresets fail, continued

By preconditioning with curvature information, AdaCore is able to magnify smaller gradient dimensions that would otherwise be ignored during coreset selection. Moreover, it

Table B.7: AdaCore outperforms other baseline subset selection algorithms as well as training on the full dataset, reaching a better accuracy in less time. This provides up to a 4.3x speedup compared to the state of the art.

	CIFAR100		Speedup over	
S = 10% R = 20	Accuracy (epoch)	Time (s)	Rand	Full
AdaCore	58.8%(200)	341	**4.3**	**2.8**
Craig	57.3%(250)	426	3.5	2.2
Random	58.1%(864)	1470	1	0.65
GradMatch	57%(200)	980	1.5	0.97
Glister	56%(300)	1110	1.3	0.86
Full Dataset	59% (40)	960	1.5	1

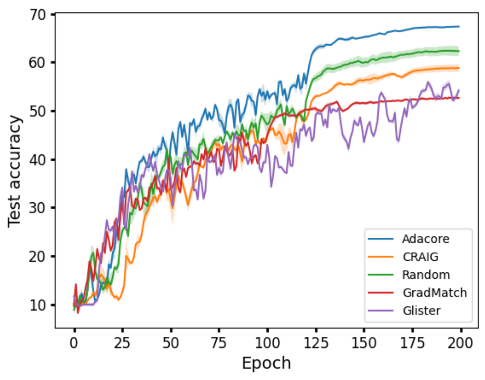
allows AdaCore to include points with similar gradients but different curvature properties. Hence, AdaCore can select more diverse subsets compared to Craig as well as GradMatch. This allows AdaCore to outperform first-order coreset methods in many regimes, such as when the subset size is large (e.g. $\geq$10%) and for larger batch size (e.g. ≥ 128).



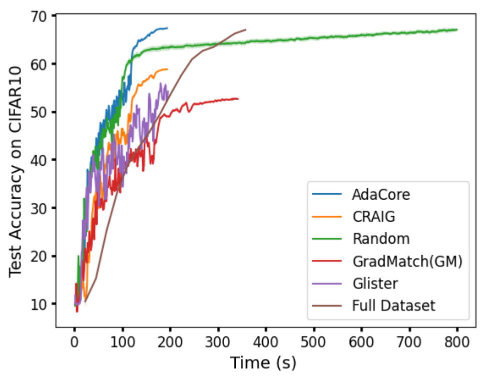
(a) AdaCore with gradient w.r.t the penultimate layer, training with SGD + Momentum

Figure B.8: Classification accuracy of ResNet20 across training on the CIFAR10 dataset, selecting coresets with AdaCore, Craig and GradMatch. Here, all coreset selection methods used the gradients of the model's last layer (dimension 64). The algorithms were calculated every $R = 20$ epoch with coreset size $S = 10\%$. Note that Craig and GradMatch are vulnerable to catastrophic forgetting, but not AdaCore.

In addition to the results shown in Figure 4.3a, (reproduced here as Fig B.9a) where $R = 1$, AdaCore outperforms Craig as well as GradMatch when we increase the coreset selection period R. Fig B.8 shows that for larger R, first-order methods succumb to



(a) Accuracy vs. Epoch



(b) Accuracy vs. Time

Figure B.9: (a) Test accuracy of AdaCore, CRAIG, Random, GradMatch and GLISTER with ResNet-18 selecting subsets of size 1% each epoch, batch size 256. (b) Training ResNet-18 on subsets of size S=1% selected every R=1 epoch, with AdaCore, Craig, Glister and GradMatch for 200 epochs vs. Random for 1000 epochs and full for 15 epochs. AdaCore outperforms baselines by providing 2x speedup over full, and more than 4.5x speedup over Random.

catastrophic forgetting each time a new subset is chosen, whereas AdaCore achieves a smooth rise in classification accuracy. This increased stability between coresets is another benefit of AdaCore's greater selection diversity. Interestingly, AdaCore achieves higher final test accuracy while selecting a smaller fraction of data points to train on during the training than Craig. Note that since AdaCore takes curvature into account while selecting the coresets, it can successfully select data points with a similar gradient but different curvature properties and extract a more diverse set of data points than Craig. However, as the coresets found by AdaCore provide a close estimation of the full preconditioned gradients for several epochs during training, the number of distinct data points selected by AdaCore is smaller than Craig.

For completeness, we provide Fig B.9b, in which we allow training random subset selection 1000 epochs. We see that it takes over 4.5x longer for Random to near the accuracy of ResNet18 trained with AdaCore and Full. We use the same experimental setup as seen in Fig B.9a.

B.2.6 MNIST

For our MNIST classifier, we use a fully-connected hidden layer of 100 nodes and ten softmax output nodes; sigmoid activation and L2 regularization with $\mu = 10^{-4}$ and mini-batch size of 32 on the MNIST dataset of handwritten digits containing 60,000 training and 10,000 test images all normalized to [0,1] by division with 255. We apply SGD with a momentum of 0.9 to subsets of size 40% of the dataset chosen at the beginning of each epoch found by AdaCore, CRAIG, and random. Fig B.10 compares the training loss and test accuracy of the network trained on coresets chosen by AdaCore, CRAIG, and random, with that of the entire dataset. We see that AdaCore can benefit from the second-order information and effectively finds subsets that achieve superior performance to that of baselines and the entire dataset. At the same time, it achieves a 2.5x speedup over training on the entire dataset.

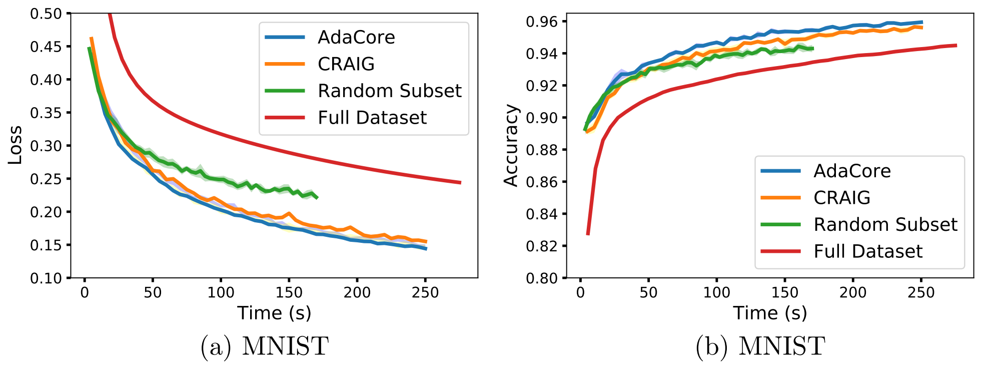


Figure B.10: Test accuracy and training loss of SGD with momentum applied to subsets found by AdaCore vs. CRAIG, and random subsets on MNIST with a 2-layer neural network. AdaCore achieves 2.5x speedup and better test accuracy, compared to training on full dataset.

B.2.7 How batch size affects coreset performance

We see in Table B.8 that training with larger batch size on subsets selected by AdaCore can achieve a superior accuracy. We reproduce Table 4.3 here with standard deviation values.

Table B.8: Training ResNet18 with S=1% subsets every R=1 epoch from CIFAR10 using batch size b= 512, 256, 128. AdaCore can leverage larger mini-bath size and obtain a larger accuracy gap to Craig and Random. For b=512, we have 1 mini-batch (GD).

	AdaCore	Craig	Rand	Gap/ Craig	Gap/ Rand
GD b=512	58.32% ± 0.45	56.32% ± 0.32	49.14% ± 1.19	1.69%	8.91%
SGD b=256	68.23% ± 0.2	58.3% ± 1.38	60.7% ± 1.04	9.93%	8.16%
SGD b=128	66.89% ± 0.73	58.17% ± 1.34	65.46% ± 0.93	8.81%	1.52%

B.2.8 Potential Social Impacts

Regarding social impact, our coreset method can outperform other methods in accuracy while selecting fewer data points over training and providing over 2.5x speedup. This will allow for a more efficient learning pipeline resulting in a lesser environmental impact. Our method can significantly decrease the financial and environmental costs of learning from big data. The financial costs are due to expensive computational resources, and environmental costs are due to the substantial energy consumption and the produced carbon footprint.

References

[AD19] Hilal Asi and John C Duchi. "The importance of better models in stochastic optimization." *arXiv preprint arXiv:1903.08619*, 2019.

[ALS15] Guillaume Alain, Alex Lamb, Chinnadhurai Sankar, Aaron Courville, and Yoshua Bengio. "Variance reduction in sgd by distributed importance sampling." *arXiv preprint arXiv:1511.06481*, 2015.

[APF] Alexander A. Alemi, Ben Poole, Ian Fischer, Joshua V. Dillon, Rif A. Saurous, and Kevin Murphy. "Fixing a Broken ELBO.".

[ARS17] Alessandro Achille, Matteo Rovere, and Stefano Soatto. "Critical Learning Periods in Deep Neural Networks." *CoRR*, **abs/1711.08856**, 2017.

[AYS16] Zeyuan Allen-Zhu, Yang Yuan, and Karthik Sridharan. "Exploiting the structure: Stochastic gradient methods using raw clusters." In *Advances in Neural Information Processing Systems*, pp. 1642–1650, 2016.

[BCG19] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A Raffel. "Mixmatch: A holistic approach to semi-supervised learning." In *Advances in Neural Information Processing Systems*, pp. 5049–5059, 2019.

[BCP16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. "Openai gym." *arXiv preprint arXiv:1606.01540*, 2016.

[Ber82] Dimitri P Bertsekas. "Projected Newton methods for optimization problems with simple constraints." *SIAM Journal on control and Optimization*, **20**(2):221–246, 1982.

[BGL21] Xiaoyu Bie, Laurent Girin, Simon Leglaive, Thomas Hueber, and Xavier Alameda-Pineda. "A Benchmark of Dynamical Variational Autoencoders applied to Speech Spectrogram Modeling." *CoRR*, **abs/2106.06500**, 2021.

[BJP22] Sunay Bhat, Jeffrey Jiang, Omead Pooladzandi, and Gregory Pottie. "De-Biasing Generative Models using Counterfactual Methods." *arXiv preprint arXiv:2207.01575*, 2022.

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer Normalization.", 2016.

[BKS07] C. Bekas, E. Kokiopoulou, and Y. Saad. "An estimator for the diagonal of a matrix." *Applied Numerical Mathematics*, **57**(11):1214–1229, 2007. Numerical Algorithms, Parallelism and Applications (2).

[BLN95] Richard H Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. "A limited memory algorithm for bound constrained optimization." *SIAM Journal on scientific computing*, **16**(5):1190–1208, 1995.

[BMB19] Vighnesh Birodkar, Hossein Mobahi, and Samy Bengio. "Semantic Redundancies in Image-Classification Datasets: The 10% You Don't Need." *arXiv preprint arXiv:1901.11409*, 2019.

[BV04a] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[BV04b] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, USA, 2004.

[CDS01] Scott Shaobing Chen, David L Donoho, and Michael A Saunders. "Atomic decomposition by basis pursuit." *SIAM review*, **43**(1):129–159, 2001.

[CG19] Yuan Cao and Quanquan Gu. "Generalization Error Bounds of Gradient Descent for Learning Over-parameterized Deep ReLU Networks.", 2019.

[CYM20] C Coleman, C Yeh, S Mussmann, B Mirzasoleiman, P Bailis, P Liang, J Leskovec, and M Zaharia. "Selection via Proxy: Efficient Data Selection for Deep Learning." In *International Conference on Learning Representations (ICLR)*, 2020.

[DB19] Nikolaos Dionelis and Mike Brookes. "Modulation-Domain Kalman Filtering for Monaural Blind Speech Denoising and Dereverberation." *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, **27**(4):799–814, 2019.

[DBK20] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale." *CoRR*, **abs/2010.11929**, 2020.

[DDS09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "ImageNet: A large-scale hierarchical image database." In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

[Den12] Li Deng. "The mnist database of handwritten digit images for machine learning research." *IEEE Signal Processing Magazine*, **29**(6):141–142, 2012.

[DHS11] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research*, **12**(7), 2011.

[Don06] David L Donoho. "Compressed sensing." *IEEE Transactions on information theory*, **52**(4):1289–1306, 2006.

[DSG20] Tri Dao, Nimit S. Sohoni, Albert Gu, Matthew Eichhorn, Amit Blonder, Megan Leszczynski, Atri Rudra, and Christopher Ré. "Kaleidoscope: an efficient, learnable representation for all structured linear maps." In *International Conference on Learning Representations (ICLR) 2020*. OpenReview.net, 2020.

[DVB15] Y. N. Dauphin, H. Vries, and Y. Bengio. "Equilibrated adaptive learning rates for non-convex optimization." In *NIPS*, pp. 1504–1512. MIT Press, 2015.

[EKD18] Ethan R Elenberg, Rajiv Khanna, Alexandros G Dimakis, Sahand Negahban, et al. "Restricted strong convexity implies weak submodularity." *Annals of Statistics*, **46**(6B):3539–3568, 2018.

[FBD09] Cédric Févotte, Nancy Bertin, and Jean-Louis Durrieu. "Nonnegative Matrix Factorization with the Itakura-Saito Divergence: With Application to Music Analysis." *Neural Computation*, **21**(3):793–830, 03 2009.

[GC19] Aaron Gokaslan and Vanya Cohen. "OpenWebText Corpus.", 2019.

[Gib14] Andrew Gibiansky. "Hessian Free Optimization.", 2014.

[GKS18] Vineet Gupta, Tomer Koren, and Yoram Singer. "Shampoo: Preconditioned stochastic tensor optimization." In *International Conference on Machine Learning*, pp. 1842–1850. PMLR, 2018.

[GL83] Daniel W. Griffin and Jae S. Lim. "Signal estimation from modified short-time Fourier transform." In *ICASSP*, 1983.

[GRB20] Donald Goldfarb, Yi Ren, and Achraf Bahamou. "Practical Quasi-Newton Methods for Training Deep Neural Networks." *CoRR*, **abs/2006.08877**, 2020.

[GRH19] Laurent Girin, Fanny Roche, Thomas Hueber, and Simon Leglaive. "Notes on the use of variational autoencoders for speech and audio spectrogram modeling."

In *DAFx 2019 - 22nd International Conference on Digital Audio Effects*, pp. 1–8, Birmingham, United Kingdom, September 2019.

[GWJ19] Will Grathwohl, Kuan-Chieh Wang, Joern-Henrik Jacobsen, David Duvenaud, Mohammad Norouzi, and Kevin Swersky. "Your classifier is secretly an energy based model and you should treat it like one." *arXiv preprint arXiv:1912.03263*, 2019.

[GZ19] Amirata Ghorbani and James Zou. "Data shapley: Equitable valuation of data for machine learning." In *International Conference on Machine Learning*, pp. 2242–2251. PMLR, 2019.

[HLL15] Thomas Hofmann, Aurelien Lucchi, Simon Lacoste-Julien, and Brian McWilliams. "Variance reduced stochastic gradient descent with neighbors." In *Advances in Neural Information Processing Systems*, pp. 2305–2313, 2015.

[HS97] S. Hochreiter and J. Schmidhuber. "Long short-term memory." *Neural Computation*, **9**(8):1735–1780, 1997.

[HSS12] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent." *Cited on*, **14**(8), 2012.

[HYL18] Lei Huang, Dawei Yang, Bo Lang, and Jia Deng. "Decorrelated Batch Normalization." *CoRR*, **abs/1804.08450**, 2018.

[HZR16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[IS15] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating

Deep Network Training by Reducing Internal Covariate Shift." *CoRR*, **abs/1502.03167**, 2015.

[JPB22] Jeffrey Jiang, Omead Pooladzandi, Sunay Bhat, and Gregory Pottie. "Hypothesis Testing using Causal and Causal Variational Generative Models." *arXiv preprint arXiv:2210.11275*, 2022.

[KAA22] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. "Elucidating the Design Space of Diffusion-Based Generative Models.", 2022.

[Kar22] Andrej Karpathy. "NanoGPT: Small GPT Implementations." `https://github.com/karpathy/nanoGPT`, 2022. Accessed on 22 February 2023.

[KB14] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980*, 2014.

[KB15] D. P. Kingma and J. L. Ba. "Adam: a method for stochastic optimization." In *ICLR*. Ithaca, NY: arXiv.org, 2015.

[KB17] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization.", 2017.

[KBC13] Anastasios Kyrillidis, Stephen Becker, Volkan Cevher, and Christoph Koch. "Sparse projections onto the simplex." In *International Conference on Machine Learning*, pp. 235–243. PMLR, 2013.

[KF18] Angelos Katharopoulos and François Fleuret. "Not all samples are created equal: Deep learning with importance sampling." In *International conference on machine learning*, pp. 2525–2534. PMLR, 2018.

[Kri09] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images." pp. 32–33, 2009.

[KSM21] Krishnateja Killamsetty, Durga Sivasubramanian, Baharan Mirzasoleiman, Ganesh Ramakrishnan, Abir De, and Rishabh Iyer. "GRAD-MATCH: A Gradient Matching Based Data Subset Selection for Efficient Learning." *arXiv preprint arXiv:2103.00123*, 2021.

[KSR20] Krishnateja Killamsetty, Durga Sivasubramanian, Ganesh Ramakrishnan, and Rishabh Iyer. "GLISTER: Generalization based Data Subset Selection for Efficient and Robust Learning." *arXiv preprint arXiv:2012.10630*, 2020.

[KW13a] Diederik P Kingma and Max Welling. "Auto-Encoding Variational Bayes.", 2013.

[KW13b] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes." *arXiv preprint arXiv:1312.6114*, 2013.

[KZR18] Kevin Kilgour, Mauricio Zuluaga, Dominik Roblek, and Matthew Sharifi. "Fr\'echet Audio Distance: A Metric for Evaluating Music Enhancement Algorithms." *arXiv preprint arXiv:1812.08466*, 2018.

[LAG19] Simon Leglaive, Xavier Alameda-Pineda, Laurent Girin, and Radu Horaud. "A Recurrent Variational Autoencoder for Speech Enhancement." *CoRR*, **abs/1910.10942**, 2019.

[LC10] Yann LeCun and Corinna Cortes. "MNIST handwritten digit database." 2010.

[Lee13] Dong-Hyun Lee. "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks." In *Workshop on challenges in representation learning, ICML*, volume 3, 2013.

[LH15] Ilya Loshchilov and Frank Hutter. "Online batch selection for faster training of neural networks." *arXiv preprint arXiv:1511.06343*, 2015.

[Li15] Xi-Lin Li. "Preconditioned Stochastic Gradient Descent.", 2015.

[Li18a] Xilin Li. "Online Second Order Methods for Non-Convex Stochastic Optimizations.", 2018.

[Li18b] Xilin Li. "Preconditioned Stochastic Gradient Descent." *IEEE Transactions on Neural Networks and Learning Systems*, **29**(5):1454–1466, 2018.

[Li18c] Xilin Li. "Preconditioner on Matrix Lie Group for SGD.", 2018.

[Li19] X. L. Li. "Preconditioner on matrix Lie group for SGD." In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[Li22] Xilin Li. "Black Box Lie Group Preconditioners for SGD." *arXiv preprint arXiv:2211.04422*, 2022.

[LZB20] Chaoyue Liu, Libin Zhu, and Mikhail Belkin. "Toward a theory of optimization for over-parameterized systems of non-linear equations: the lessons of deep learning." *arXiv preprint arXiv:2003.00307*, 2020.

[Mar16] James Martens. *Second-order Optimization for Neural Networks*. PhD thesis, University of Toronto, Canada, 2016.

[MBK15] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrák, and Andreas Krause. "Lazier than lazy greedy." In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[MBL19] Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. "Coresets for Data-efficient Training of Machine Learning Models." *arXiv preprint arXiv:1906.01827*, 2019.

[MBL20] Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. "Coresets for data-efficient training of machine learning models." In *International Conference on Machine Learning*, pp. 6950–6960. PMLR, 2020.

[MG76] John D. Markel and Augustine H. Gray. *Vocoders*, pp. 227–262. Springer Berlin Heidelberg, Berlin, Heidelberg, 1976.

[MG15a] J. Martens and R. B. Grosse. "Optimizing neural networks with Kronecker-factored approximate curvature." In *ICML*, pp. 2408–2417, 2015.

[MG15b] James Martens and Roger Grosse. "Optimizing neural networks with kronecker-factored approximate curvature." In *International conference on machine learning*, pp. 2408–2417, 2015.

[Min78] Michel Minoux. "Accelerated greedy algorithms for maximizing submodular set functions." In *Optimization techniques*, pp. 234–243. Springer, 1978.

[MIY14] Akira Maezawa, Katsutoshi Itoyama, Kazuyoshi Yoshii, and Hiroshi G. Okuno. "Nonparametric Bayesian Dereverberation of Power Spectrograms Based on Infinite-Order Autoregressive Processes." *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, **22**(12):1918–1930, 2014.

[MKS13] Baharan Mirzasoleiman, Amin Karbasi, Rik Sarkar, and Andreas Krause. "Distributed submodular maximization: Identifying representative elements in massive data." In *Advances in Neural Information Processing Systems*, pp. 2049–2057, 2013.

[MMK18] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. "Virtual adversarial training: a regularization method for supervised and semi-supervised learning." *IEEE transactions on pattern analysis and machine intelligence*, **41**(8):1979–1993, 2018.

[MMS17] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. "Towards deep learning models resistant to adversarial attacks." *arXiv preprint arXiv:1706.06083*, 2017.

[MS12] J. Martens and I. Sutskever. "Training deep and recurrent neural networks with Hessian-free optimization." In G. Montavon, G. B. Orr, and K. R. Muller, editors, *Neural Networks: Tricks of the Trade*. Springer, Berlin Heidelberg, 2012.

[MSM93] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. "Building a Large Annotated Corpus of English: The Penn Treebank." *Computational Linguistics*, **19**(2):313–330, 1993.

[Nak20] Toru Nakashika. "Complex-Valued Variational Autoencoder: A Novel Deep Generative Model for Direct Representation of Complex Spectra." In *INTERSPEECH*, 2020.

[Nat95] Balas Kausik Natarajan. "Sparse approximate solutions to linear systems." *SIAM journal on computing*, **24**(2):227–234, 1995.

[NPH20] Erik Nijkamp, Bo Pang, Tian Han, Linqi Zhou, Song-Chun Zhu, and Ying Nian Wu. "Learning multi-layer latent variable model via variational optimization of short run MCMC for approximate inference." *stat*, **1050**:17, 2020.

[NWC11] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. "Reading digits in natural images with unsupervised feature learning." 2011.

[OIY22] Kazuki Osawa, Satoki Ishikawa, Rio Yokota, Shigang Li, and Torsten Hoefler. "ASDL: A Unified Interface for Gradient Preconditioning in PyTorch." In *Order Up! The Benefits of Higher-Order Optimization in Machine Learning, NeurIPS 2022 Workshop*, 2022.

[PDM22a] Omead Pooladzandi, David Davini, and Baharan Mirzasoleiman. "Adaptive Second Order Coresets for Data-efficient Machine Learning." In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 17848–17869. PMLR, 17–23 Jul 2022.

[PDM22b] Omead Pooladzandi, David Davini, and Baharan Mirzasoleiman. "Adaptive Second Order Coresets for Data-efficient Machine Learning." In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 17848–17869. PMLR, 17–23 Jul 2022.

[Pea94] Barak A Pearlmutter. "Fast exact multiplication by the Hessian." *Neural computation*, **6**(1):147–160, 1994.

[PEC12] Mert Pilanci, Laurent El Ghaoui, and Venkat Chandrasekaran. "Recovery of sparse probability measures via convex programming." 2012.

[PHN20] Bo Pang, Tian Han, Erik Nijkamp, Song-Chun Zhu, and Ying Nian Wu. "Learning latent space energy-based prior model." *arXiv preprint arXiv:2006.08205*, 2020.

[PJB23] Omead Pooladzandi, Jeffrey Jiang, Sunay Bhat, and Gregory Pottie. "Towards Composable Distributions of Latent Space Augmentations." *arXiv preprint arXiv:2303.03462*, 2023.

[PKN23] Omead Pooladzandi, Pasha Khosravi, Erik Nijkamp, and Baharan Mirzasoleiman. "Generating High Fidelity Synthetic Data via Coreset selection and Entropic Regularization.", 2023.

[PLG23] Omead Pooladzandi, Xilin Li, Yang Gao, and Lalin Theverapperuma. "Exploring the Potential of VAE Decoders for Enhanced Speech Re-Synthesis." In *2023 IEEE Statistical Signal Processing Workshop (SSP)*, 2023.

[PZ22] Omead Pooladzandi and Yiming Zhou. "Improving Levenberg-Marquardt Algorithm for Neural Networks." *arXiv preprint arXiv:2212.08769*, 2022.

[Qia99] Ning Qian. "On the momentum term in gradient descent learning algorithms." *Neural networks*, **12**(1):145–151, 1999.

[Raf20] Colin Raffel. "FixMatch: Simplifying Semi-Supervised Learning with Consistency and Confidence." 2020.

[RBH01] A.W. Rix, J.G. Beerends, M.P. Hollier, and A.P. Hekstra. "Perceptual evaluation of speech quality (PESQ)-a new method for speech quality assessment of telephone networks and codecs." In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*, volume 2, pp. 749–752 vol.2, 2001.

[RM51] H. Robbins and S. Monro. "A stochastic approximation method." *The Annals of Mathematical Statistics*, pp. 400–407, 1951.

[RRJ93] L. Rabiner, L.R. Rabiner, and B.H. Juang. *Fundamentals of Speech Recognition.* Prentice-Hall Signal Processing Series: Advanced monographs. PTR Prentice Hall, 1993.

[RWC19] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. "Language Models are Unsupervised Multitask Learners." 2019.

[SBL16] Levent Sagun, Léon Bottou, and Yann LeCun. "Singularity of the Hessian in Deep Learning." *CoRR*, **abs/1611.07476**, 2016.

[SDS19] Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. "Green ai." *arXiv preprint arXiv:1907.10597*, 2019.

[SEG17] Levent Sagun, Utku Evci, V. Ugur Güney, Yann N. Dauphin, and Léon Bottou. "Empirical Analysis of the Hessian of Over-Parametrized Neural Networks." *CoRR*, **abs/1706.04454**, 2017.

[SGM19] Emma Strubell, Ananya Ganesh, and Andrew McCallum. "Energy and Policy Considerations for Deep Learning in NLP." In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 3645–3650, 2019.

[SQA15] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. "Prioritized experience replay." *arXiv preprint arXiv:1511.05952*, 2015.

[SWD17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal Policy Optimization Algorithms.", 2017.

[SZL13] Tom Schaul, Sixin Zhang, and Yann LeCun. "No more pesky learning rates." In *International Conference on Machine Learning*, pp. 343–351. PMLR, 2013.

[Tib96] Robert Tibshirani. "Regression shrinkage and selection via the lasso." *Journal of the Royal Statistical Society: Series B (Methodological)*, **58**(1):267–288, 1996.

[TIY18] Hiroshi Takahashi, Tomoharu Iwata, Yuki Yamanaka, Masanori Yamada, and Satoshi Yagi. "Student-t Variational Autoencoder for Robust Density Estimation." In *IJCAI*, pp. 2696–2702, 2018.

[TSC18] Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J Gordon. "An empirical study of example forgetting during deep neural network learning." *arXiv preprint arXiv:1812.05159*, 2018.

[TSC19a] Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J Gordon. "An Empirical Study of Example Forgetting during Deep Neural Network Learning." In *ICLR*, 2019.

[TSC19b] Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J. Gordon. "An Empirical Study of Example Forgetting during Deep Neural Network Learning." In *International Conference on Learning Representations*, 2019.

[TV17] Antti Tarvainen and Harri Valpola. "Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results." In *Advances in neural information processing systems*, pp. 1195–1204, 2017.

[VWB16] Andreas Veit, Michael Wilber, and Serge Belongie. "Residual Networks Behave Like Ensembles of Relatively Shallow Networks.", 2016.

[WH18] Yuxin Wu and Kaiming He. "Group Normalization." *CoRR*, **abs/1803.08494**, 2018.

[Wol82] Laurence A Wolsey. "An analysis of the greedy algorithm for the submodular set covering problem." *Combinatorica*, **2**(4):385–393, 1982.

[WRS17] Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, and Benjamin Recht. "The Marginal Value of Adaptive Gradient Methods in Machine Learning.", 2017.

[WWR21] Zhong-Qiu Wang, Gordon Wichern, and Jonathan Le Roux. "On The Compensation Between Magnitude and Phase in Speech Separation." *CoRR*, **abs/2108.05470**, 2021.

[WZG20] Chaoqi Wang, Guodong Zhang, and Roger Grosse. "Picking winning tickets before training by preserving gradient flow." *arXiv preprint arXiv:2002.07376*, 2020.

[XRM20] Peng Xu, Fred Roosta, and Michael W Mahoney. "Second-order optimization for non-convex machine learning: An empirical study." In *Proceedings of the 2020 SIAM International Conference on Data Mining*, pp. 199–207. SIAM, 2020.

[YCW20] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. "BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning." In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[YGS20] Zhewei Yao, Amir Gholami, Sheng Shen, Kurt Keutzer, and Michael W Mahoney. "ADAHESSIAN: An adaptive second order optimizer for machine learning." *arXiv preprint arXiv:2006.00719*, 2020.

[YGS21] Zhewei Yao, Amir Gholami, Sheng Shen, Mustafa Mustafa, Kurt Keutzer, and Michael W. Mahoney. "AdaHessian: an adaptive second order optimizer for machine learning." In *AAAI*, 2021.

[YW21] Chia-Hung Yuan and Shan-Hung Wu. "Neural Tangent Generalization Attacks." In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 12230–12240. PMLR, 18–24 Jul 2021.

[YXR18] Zhewei Yao, Peng Xu, Farbod Roosta-Khorasani, and Michael W Mahoney. "Inexact non-convex Newton-type methods." *arXiv preprint arXiv:1802.06925*, 2018.

[ZBM22] Guodong Zhang, Aleksandar Botev, and James Martens. "Deep Learning without Shortcuts: Shaping the Kernel with Tailored Rectifiers." *arXiv preprint arXiv:2203.08120*, 2022.

[Zei12] Matthew D Zeiler. "Adadelta: an adaptive learning rate method." *arXiv preprint arXiv:1212.5701*, 2012.

[ZFM20] Pan Zhou, Jiashi Feng, Chao Ma, Caiming Xiong, Steven Hoi, and Weinan E. "Towards Theoretically Understanding Why SGD Generalizes Better Than ADAM in Deep Learning.", 2020.

[ZK16] Sergey Zagoruyko and Nikos Komodakis. "Wide residual networks." *arXiv preprint arXiv:1605.07146*, 2016.

[ZTD20] Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar Tatikonda, Nicha Dvornek, Xenophon Papademetris, and James S. Duncan. "AdaBelief optimizer: adapting stepsizes by the belief in observed gradients." In *NeurIPS 2020*, 2020.