Approximation approaches for training neural network problems with dynamic mini-batch sub-sampled losses

by

Younghwan Chae

This thesis is submitted in partial fulfilment of the requirements for the degree

Philosophiae Doctor (Mechanical Engineering)

in the

Faculty of Engineering, the Built Environment and Information Technology

University of Pretoria

Pretoria

South Africa

2021

To the pursuit of progress...

“Life is all about gradient, focus on your growth and learning.” - Daniel N. Wilke

Abstract

Title: Approximation approaches for solving neural network problems for dynamic mini-batch sub-sampled losses

Author: Younghwan Chae

Supervisor: Daniel N. Wilke

Learning rate schedule parameters is a sensitive and challenging hyperparameter to resolve in machine learning. It needs to be resolved whenever a model, data, data preprocessing or data batching changes. Implications of poorly resolving learning rates include poor models, high computing cost, excessive training time, and excessive carbon footprint. In addition, deep neural network (DNN) architectures routinely require billions of parameters, with GPT-3 utilizing 175 billion parameters and an estimated 12 million USD to train. Mini-batch sub-sampling introduces bias and variance that can manifest in several ways. Considering a line-search along a descent direction, the implications are smooth loss functions with large bias (static) or point-wise discontinuous loss functions with low bias but high variance in the function response. Two previous studies demonstrated that line searches have the potential to automate learning rate selection. In both cases, learning rates are resolved for point-wise discontinuous functions that include Bayesian regression and direct optimization using a gradient-only line search, GOLS. This study is an explorative study that investigates the potential of surrogates to resolve learning rates instead of direct optimization of the loss function. We aim to identify domains that warrant further investigation, for which purposes we introduced a new robustness measure to compare algorithms more sensibly. As a result, we start our surrogate investigation at the fundamental level, considering the most basic form for each approach. This isolates the essence and rids unnecessary complexity. We do, however, retain selected complexity that is deemed crucial such as dynamic sub-sampling. Hence, this study is an explorative study and not yet another study that proposes a state-of-the-art (SOTA) algorithm on a carefully curated dataset with carefully curated baseline algorithms against which to compare. The three fundamentally different approaches to resolve learning rates using surrogates are

1. The construction of one-dimensional quadratic surrogates for point-wise discontinuous functions to resolve learning rates by minimization;

2. The construction of one-dimensional classifiers to resolve learning rates from a gradient-only perspective using classification;

3. Sub-dimensional surrogates (higher than 1D) on smooth loss functions to isolate the identification of appropriate bases on simple test problems.

This study concludes that both 1 and 2 further warrant investigation, with the longer-term goal to be extended to sub-dimensional surrogates to enhance efficiency.

Acknowledgements

I would like to my extend my deepest gratitude to Prof. Wilke for all his help throughout this arduous journey. I first made Prof. Wilke's acquaintance in 2014, and until now his logical, productive, and diligent mindset has been a constant source of inspiration and respect. Through many different projects, I was privileged to receive his teaching and advice. Whilst completing my Master's and Doctor's degrees under his guidance, Prof. Wilke has pushed me to come this far, and for that, I will always be indebted to him. I have so much more to learn from him and hope he will always be my mentor. Prof. Wilke has taught me that a true researcher should never stop asking the question 'why?'. I will always keep this lesson in my heart. In this long period of working together as co-researchers I feel that I have made great progress both as a person and as a researcher, and I believe this experience will be a source of strength throughout my life.

Next, I would like to thank my colleague, Dominic Kafka. For many years while sharing the same office, we developed a great friendship. We spent many hours sharing numerous ideas and valuable thoughts, reflecting on and discussing their feasibility. I would like to thank him for giving me this opportunity to let those ideas flow and thank him for his friendship through thick and thin. I could never have reached this place on my own. I especially would like to acknowledge him for helping me in my struggle with the English language throughout my research.

Also, to Prof. Heyns as the head of the Centre for Asset and Integrity Management (C-AIM), Department of Mechanical and Aeronautical Engineering at the University of Pretoria, South Africa, thank you for providing all the resources necessary and support to complete my Doctor's degree.

Lastly, I would like to thank my family for their continued love and support throughout the years. It was difficult to be so far away but thank you for always making me feel like I am never alone. Thank you for all your prayers and emotional support. Without you all, I would not be here right now.

Without the help of the above-mentioned people in my life, it would not have been possible to attain this degree. I believe the only way to repay this heart of gratitude is to give back to the country and society what I have gained through this journey.

Contents

List of Figures

List of Tables

List of Abbreviations

MBSS	Mini-batch sub-sampling
NN-GPP	Non-Negative Associated Gradient Projection Point
SNN-GPP	Stochastic Non-Negative Associated Gradient Projection Point
SGD	Stochastic Gradient Descent algorithm
SSGD	Scheduled SGD algorithm
IAC	Initial Accept Condition
GOS	Gradient-Only Surrogate
GOALS	Gradient-Only Approximation Line Search
GOCLS	Gradient-Only (Bayesian) Classification Line Search
GOLS-I	Gradient-Only Line Search that is Inexact
DNN	Deep Neural Network
MNIST	Modified National Institute of Standards and Technology
CIFAR	Canadian Institute For Advanced Research
PLS	Probabilistic Line Search
COBRA	Constrained Optimization By Radial basis function Approximation
DYCORS	DYnamic COordinate search Response Surface models
WGEK	Weighted Gradient-Enhanced Kriging
CD	Coordinate descent
CCD	Cyclic Coordinate Descent
RCD	Random Coordinate Descent
BCD	Block Coordinate Descent
DDS	Dynamically Dimensioned Searches
GPU	Graphics Processing Unit

List of Symbols

n	Counter for iteration number
i	Counter for function evaluation within an iteration
$.^m$	An optimum solution of static MBSS
$.^*$	An optimum solution of dynamic MBSS
$\|.\|$	Cardinality of a set
α	Learning rate
$\mathcal{B}$	Uniform-randomly sub-sampled mini-batch, a subset of training set, T
M	Total number of training samples, equivalent to $\|T\|$
$\boldsymbol{d}_n$	Search direction used for an algorithm at iteration n
$\boldsymbol{t}_b$	Training dataset pair (input and output) in training set T for observation b
$\boldsymbol{x}$	Weights for neural network models
$\mathcal{L}(\boldsymbol{x})$	Deterministic full-batch loss function
$\boldsymbol{\nabla}\mathcal{L}(\boldsymbol{x})$	Deterministic full-batch gradient function
$\ell(\boldsymbol{x};\boldsymbol{t}_b)$	Individual sample loss for a training sample, $\boldsymbol{t}_b$
$\boldsymbol{\nabla}\ell(\boldsymbol{x};\boldsymbol{t}_b)$	Individual sample gradient for a training sample, $\boldsymbol{t}_b$
$\mathcal{F}(\alpha)$	1-D full-batch sampled loss for a specified search direction, $\boldsymbol{d}_n$
$\mathcal{F}'(\alpha)$	1-D full-batch sampled directional derivative function for a specified search direction, $\boldsymbol{d}_n$
$\bar{.}$	Static MBSS function evaluated using $\mathcal{B}_n$
$\tilde{.}$	Dynamic MBSS function evaluated using $\mathcal{B}_{n,i}$
$\hat{.}$	Approximation model function
$L(\boldsymbol{x})$	Mini-batch loss function
$\boldsymbol{g}(\boldsymbol{x})$	Mini-batch gradient function
$f(\alpha)$	1-D loss function for a specified search direction, $\boldsymbol{d}_n$
$f'(\alpha)$	1-D directional derivative function for a specified search direction, $\boldsymbol{d}_n$
$\boldsymbol{k}$	Coefficients for $\hat{f}(\alpha)$ and $\hat{f}'(\alpha)$
$\boldsymbol{A}$	Spatial location matrix
$\boldsymbol{b}$	Vector containing loss and/or directional derivatives, where $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{k}$
ε	Prescribed tolerance
$\mathcal{N}_{input}$	Number of weights in the input layer
$\mathcal{N}_{output}$	Number of weights in the output layer
B_ϵ	Ball containing all SNN-GPPs
ω	Prescribed Armijo constant
c	Curvature hyperparameter
c_1	Undershooting hyperparameter
c_2	Overshooting hyperparameter

γ	Default learning rate for optimizers
$\mathcal{I}$	Learning rate interval for bracketing strategy
$\cdot_L$	Value corresponds to the lower bound of interval $\mathcal{I}$
$\cdot_U$	Value corresponds to the upper bound of interval $\mathcal{I}$
$\Gamma(\boldsymbol{x})$	Lyapunov function
$\cdot_\xi$	Positive directional derivative sign
$\cdot_\zeta$	Negative directional derivative sign
π	Desired probability of observing a directional derivative sign
$\hat{\pi}$	Sample probability of observing a directional derivative sign
A	Set of directional derivative signs
S	Set of different sign classes (positive and negative)
μ	Mean value obtained from a distribution
σ^2	Variance obtained from distribution
$\mathcal{N}(\mu, \sigma^2)$	Normal distribution with mean of μ and variance of σ^2
μ_e	Mean value for an exploration function
σ_e^2	Variance for an exploration function
σ_a^2	Overall variance
α_f	Final learning rate on the $\log_{10}$ scale
w	Original problem dimension
v	Reduced problem dimension, where $v \ll w$
$\boldsymbol{D}$	Original bound constrained domain
$\boldsymbol{D}_s$	Sampling domain for a surrogate model
$\boldsymbol{N}$	Scaled bound constrained domain
$\boldsymbol{z}$	Weights in scaled bound constrained domain, $\boldsymbol{z} \in \boldsymbol{N}$
$\boldsymbol{\Phi}$	Cubic radial basis function matrix
r	Radial basis function
$\boldsymbol{c}$	Polynomial coefficients
ϕ	Cubic radial basis function
$\boldsymbol{\lambda}$	Weights for radial basis function
$\boldsymbol{P}$	Low order polynomial matrix
$\boldsymbol{x}^{v*}$	Sub-dimensional minimizer in $\boldsymbol{D}_s$
$\boldsymbol{z}^{v*}$	Sub-dimensional minimizer in $\boldsymbol{N}$
$\boldsymbol{Q}$	Arbitrary proper orthogonal matrix
p	Number of data points evaluated for constructing a model
τ	Scaling factor

Chapter 1

Overview

Advancements in computer technologies made machine learning implementable to assist our daily lives in many forms. However, the difficulty of solving machine learning optimization problems is overgrowing due to an extension in the size and complexity of models [Goodfellow et al., 2016, Brown et al., 2020]. No matter how accurately a model is developed, the model cannot convincingly perform if it is not adequately trained. Hence, we must not neglect the importance of improving optimization techniques concurrent with refining machine learning models. This thesis mainly aims to discover various potential training optimization methods that have not been attempted elsewhere. Therefore, we do not expect any State of the Arts (SOTA) performances, but this thesis should build the foundation for various novel optimization strategies, specifically for deep learning.

1.1 Training strategies for DNNs

Regarding deep neural network (DNN) problems, numerous strategies exist to improve training and test performances as shown in Figure 1.1. Popular methods concerning data include increasing data size, data augmentation [Shorten and Khoshgoftaar, 2019], data rescaling, and data transformation [Bishop, 2006]. Concerning optimization, regularization techniques [Kukačka et al., 2017] are used to prevent overfitting, various weight initialization [Boulila et al., 2021] and activation functions [Nwankpa et al., 2018] may improve the training results. Mini-batch sub-sampling methods [Csiba and Richtárik, 2018] reduce the computational cost by limiting the memory usage while it produces a better generalization effect [Masters and Luschi, 2018]. An appropriate choice of loss functions defines the objectives of the problem, and optimizers help adjust the weights of the problem to satisfy the objectives.

1.2 Various optimizers for DNNs

Among many strategies to enhance training DNNs, this research focuses on optimizers. The most straightforward yet most robust optimizer is the stochastic gradient descent (SGD) algorithm. For better performance, SGD with momentum [Queipo et al., 2005] and Nesterov accelerated gradient (NAG) [Sutskever et al., 2013] are developed. Nevertheless, these gradient-based algorithms remain challenging when choosing the learning rates. The adaptive learning rate methods such as Adagrad [Duchi et al., 2011], Adadelta [Zeiler, 2012b], RMSProp [Tieleman and Hinton, 2012], Adam [Kingma and Ba, 2014], and the variations of Adam, AdamW [Loshchilov and Hutter, 2017b], AdaMax, and Nadam [Dozat, 2016] are developed. However, it does not mean that their prespecified learning rates are entirely insensitive to the results, and the recommended learning rates exist for each optimizer. Besides, gradient-based adaptive algorithms may generalize worse than SGD, although they accelerate the training speed [Zhou et al., 2020].

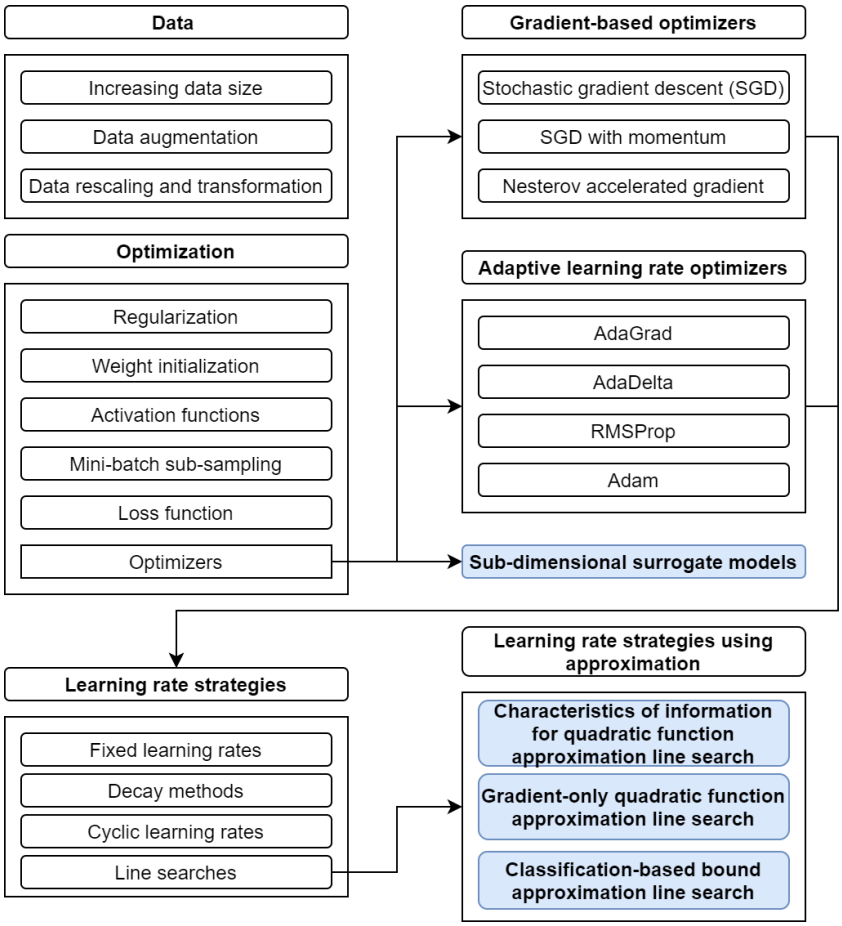


Figure 1.1: Illustration of strategies for improving training results in DNNs. The colored cells indicate the contributions of the thesis.

1.3 Adopting surrogate models for training DNNs

Learning rate is perhaps the most critical and challenging hyperparameter to select for practical training results because almost any choices we make may alter the characteristics of problems. These include all the strategies mentioned earlier, including the selection of optimizers. Furthermore, DNNs are often computationally costly to evaluate the functions, high-dimensional, black-box, and even discontinuous for specific mini-batch sampling methods [Kafka and Wilke, 2019a].

Interestingly, surrogate models (or approximation models for low dimension) are known for their computational efficiency for optimizing 1) computationally expensive, 2) black-box, 3) discontinuous functions since it only requires much fewer function evaluations to construct a smooth continuous model without identifying the actual mathematical formulation. The only critical downside to using surrogate models for training DNNs is the "curse of dimensionality," which exponentially increases the computational requirements for the high-dimensional problem. However, the excellent news is that we only require one dimensional model for line searches, determining optimal learning rates for the descent directions. This motivates our aim to discover potential approaches for adopting surrogate models in training DNNs.

1.4 Rationale of chapters

The thesis largely consists of two parts, as shown in 1.2. Specifically, we split the complexities relating to surrogate dimensionality and surrogate surface estimation given high variance data. This is done to avoid drawing poor conclusions as a result of interaction between the two. Hence, the thesis mainly consists of two parts. In Part 1, we eliminate complexities related to dimensionality but embrace the complexities arising from high variance in the data to establish which information should be considered to approximate the function. Hence, in Part 1, we only consider 1D surrogates. In Part 2, we eliminate the complexities of variance in the data but embrace the complexities arising from moving beyond 1D to higher subdimensions. Hence, we only consider smooth and continuous loss functions, largely based on well-known test problems.

Part 1 develops surrogate-based line searches using various deterministic and stochastic 1-D approximation models. Investigating the basic 1-D approximations allows us to closely explore mini-batch sub-sampling (MBSS) implications, manifesting as biases and variances in functions. While static MBSS yields large biases and small variances, dynamic MBSS produces large variances and small biases. We utilize dynamic MBSS in Part 1 because it allows more significant data throughput and yields less bias. Besides, two recent studies introduced line searches in the dynamic MBSS setting using the Bayesian regression model [Mahsereci and Hennig, 2017] and the direct optimization approach using only directional derivative sign information [Kafka and Wilke, 2019a]. This motivates us to investigate approximation-based line searches that reduce model error exhibited as the variance. In addition, we limit the model complexities to the simplest model that can be deployed to allow us to focus on which information, function values, or directional derivative to consider for the surrogate approximation of point-wise discontinuous loss functions.

Part 2 focuses on sub-dimensional surrogate-based optimization, in short, subsurrogate, which are higher-dimensional models. Part 2, in contrast to Part 1, builds up the complexity of the model while the information setting is as simple as possible using a compact neural network with full-batch sampling, which removes nonlinearity in functions and discontinuities from the loss function. This allows us to investigate the models' actual behavior rather than its response to additional complexities. Utilizing the high-dimensional model as an optimizer has the benefit of expressing the complexity of the original problem in more detail and adjust the search directions within the subspace.

We explore three distinct approximation approaches, the first two in Part 1 and the last in Part 2, in the thesis:

- Chapter 2 and 3: Minimization approach using deterministic quadratic function approxi-

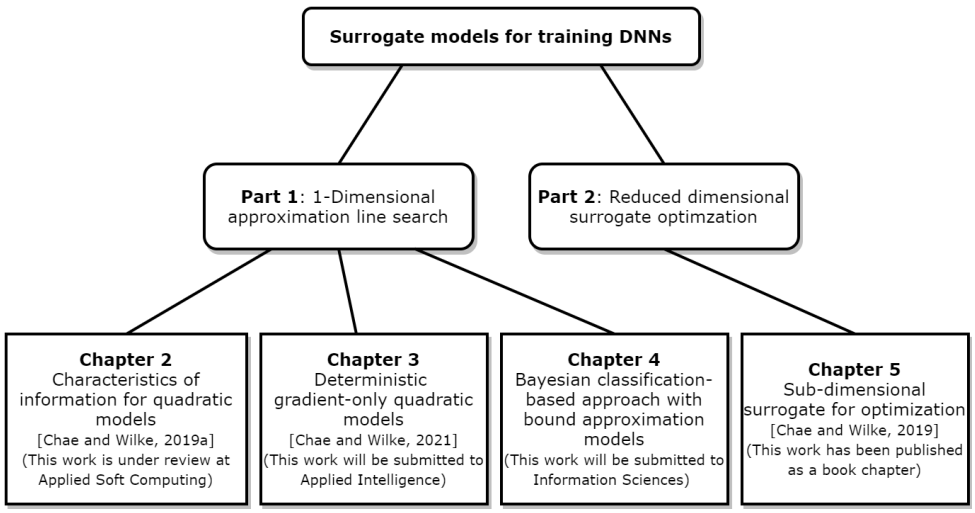


Figure 1.2: The outline of the main chapters in the thesis.

mations;

- Chapter 4: Bayesian classification approach using directional derivative sign bound approximations;
- Chapter 5: Minimization approach using sub-dimensional surrogate models.

1.4.1 Part 1: Line searches with 1-D approximation models

Part 1 contains three chapters, Chapters 2-4, and aims to investigate the potential for various deterministic and stochastic approximations as line searches. The three chapters commonly adopt the dynamic mini-batch sub-sampling (MBSS). The mini-batch is uniform-randomly resampled within every function evaluation in iteration, creating stochastic and discontinuous function settings. Dynamic MBSS has the benefit of allowing faster data throughput.

Chapter 2: "Empirical Study Towards Understanding Line Search Approximations For Training Deep Neural Networks" [Chae and Wilke, 2019a]

Chapter 2 studies the characteristics of five distinct deterministic quadratic approximation models built using different information, including function values, directional derivatives, and their mixtures. Quadratic models are chosen because they are the most simplistic deterministic models with local minima in 1-D. This work aims to study the relationships between the information type and the approximation errors, which is crucial for developing line searches. We realize that both spatial location and the number of information pieces used for the models have a diverse effect on the approximation errors. Involving a lot of unnecessary information may increase the error, but directional derivative information at the origin of every approximation minimizes the variance in the models. The study showed that one of the most promising five quadratic approximations was the gradient-only surrogate (GOS) which we continued to work on in the following chapter.

Chapter 3: "GOALS: Gradient-Only Approximations for Line Searches Towards Robust and Consistent Training of Deep Neural Networks" [Chae et al., 2021]

Chapter 3 extends the GOS model to a robust line search algorithm by adopting the Wolfe convergence criteria. Our proposed line search, so-called Gradient-Only Approximation Line Search (GOALS), sequentially builds gradient-only approximations until the convergence criteria meet. With curvature condition hyperparameter, it allows the users to trade between the precision and computational cost and select the degrees of either undershooting or overshooting. However,

the disadvantage of GOALS is that it may produce a significant approximation error if one of the spatial locations of information for building approximation models is too far or too close to each other.

Recall that we intend to show potentials, not optimal performance, in each approach that we present. Hence, we introduced a new concept evaluation criteria called relative robustness measure to evaluate our strategies. While the conventional performance measure only considers the top performances, such as the Top-1 or Top-5 accuracies, the relative robustness measure also considers the poor performances to measure each strategy's consistency across different problems. We implement this measure in Chapters 3 and 4 to choose hyperparameters and compare our algorithms against other methods.

Based on the relative measure, we found out that the distance between each data point for constructing approximations needs to be shorter for more complex DNN problems than shallower DNN problems to reduce the model errors. Due to the convergence criteria adopted, GOALS tends to be less aggressive in terms of performance compared to vanilla GOS. However, it ranked third and second among the ten strategies we compared for training and test robustness measures to show its potential for future research.

Chapter 4: "Gradient-Only Line Search With Bayesian Classication Approach For Training Neural Networks"

Line searches often require more than one function evaluation to find a learning rate that satisfies the objective function. This can easily be costly when the function is computationally expensive to evaluate. Chapter 4 introduces a Bayesian classification-based line search (GOCLS) using the bound approximations. While the previous two chapters used the deterministic quadratic models, which required extra gradient computations for descent directions, Gradient-Only Classification Line Search (GOCLS) uses the historical directional derivative data from previous iterations. It only requires one gradient evaluation per iteration. Using the historical data, GOCLS builds two probability distributions for positive and negative signs of directional derivatives. It predicts the learning rates at which the signs change from negative to positive at a user-specified target probability. The target probability is a comprehensible hyperparameter. It trades between undershooting and overshooting by setting it below or above 50%, respectively. We also perform an investigation to tune this hyperparameter for different optimizers such as SGD, RMSProp, and Adam.

This chapter again restates the relative robustness measure, and based on the measure, the robustness of GOCLS is compared against other learning rate strategies. The results show that it ranks third for both training and test robustness measures, leading the GOALS algorithm across different problems. Hence, the results of GOCLS support the potential of the classification-based approach for future researches.

Transition from Part 1 to Part 2

Part 1 investigates the potential for the various approximations for line searches which is 1-D. Therefore, We could build the complexity of the problems by adopting gradient-only models, dynamic MBSS causing point-wise discontinuity to the problem. However, Part 2 studies the potential for reduced sub-dimensional surrogate (or subsurrogate) models in higher dimensions than Part 1 for training DNN problems. Hence, we intentionally keep the problem settings as simple as possible to add clarity to the model's behaviors. This means that we no longer use a gradient-only model but a conventional function-value-only model and not dynamic mini-batch but full-batch for a single hidden layer neural network problems.

1.4.2 Part 2: Optimization with sub-dimensional surrogate models

Part 2 consists of one chapter introducing a novel crude concept of optimization technique that adopts multi-dimensional subsurrogate. Its dimension is much less than the original problem but higher than one (e.g., line searches).

Chapter 5: "Sub-dimensional Surrogates to Solve High Dimensional Optimization Problems in Machine Learning" [Chae and Wilke, 2019b]

Chapter 5 investigates the potential for multi-dimensional subsurrogate (or sub-dimensional surrogate) models for training NNs in the most basic neural network problem settings. Based on the sensitivity ranked by the magnitudes of the gradient, we test the various number of the weights to build a cubic radial basis function (RBF) surrogate model with a linear term for avoiding collinearity. In every iteration, these computational inexpensive low dimensional models are updated. As we optimize the weights involved in the model by minimizing the function value, the resolved weights are updated in the original problem. This process is repeated until it reaches convergence.

We performed hyperparameter studies, including varying the models' dimensionality, data points, data sampling domains, and selection methods for the weights. The results show that the higher the surrogate dimension is, the faster the convergence becomes, but it also requires more data point and function evaluations. Hence, reducing the dimensionality and the data points while training helps for faster convergence. The rate of reducing the sampling domain turns out to be trading between localized search and wasting computational resources. However, although complexity follows this approach as many hyperparameters to choose from, we show that this method falls within the potential area for future researches.

Chapter 2

Empirical Study Towards Understanding Line Search Approximations For Training Deep Neural Networks

2.1 Summary of chapter

Selecting appropriate learning rates is a critical task in training deep neural networks (DNNs) for reducing the computational cost. Concerning line searches, keeping the mini-batch size small helps to enhance the training speed. Employing dynamic mini-batch sub-sampling (MBSS), which refers to sub-sampling mini-batches for every loss function evaluation, helps to increase the data throughput as compared to static MBSS, which sub-samples only when the search direction updates. However, dynamic MBSS loss functions are point-wise discontinuous or stochastic. This study investigates the influence of enforcing different pieces of information (function values or directional derivatives) at different spatial locations along a search direction on the quality of univariate line search approximations. In particular, our empirical investigation focuses on quadratic approximations to approximate stochastic loss functions. We examine five quadratic approximation models with function and directional derivative information enforced in various ways. The results for several neural network problems show that enforcing the directional derivative information at the origin helps to reduce the variance of predicted learning rates and that using less but specific data reduces approximation errors. The study demonstrates that quadratic approximations could potentially be proposed to resolve learning rates for point-wise discontinuous loss functions automatically.

2.2 Introduction

Training neural networks on large training sets remain challenging as the effectiveness thereof is determined by learning rates (or step sizes), that are not known upfront. Factors such as the optimizer, data set, scaling, and network architecture can significantly influence what is useful. During these difficulties, we need to search for efficient learning rates with special care [Bengio, 2012, Smith, 2017], in particular for large-scale problems, because choosing an inefficient learning rate may significantly expand the required computational cost, and that then results in energy waste causing unnecessary CO_2 emissions [Strubell et al., 2019].

Choosing learning rates is further complicated when having to resolve them within a stochastic optimization setting [Robbins and Monro, 1951] because sampling errors are introduced by mini-batch sub-sampling (MBSS). A mini-batch can be updated for every loss function evaluation, often referred to as dynamic MBSS, or only when the search direction is updated, referred to as static MBSS. For the latter, the mini-batch, therefore, remains fixed for loss function eval-

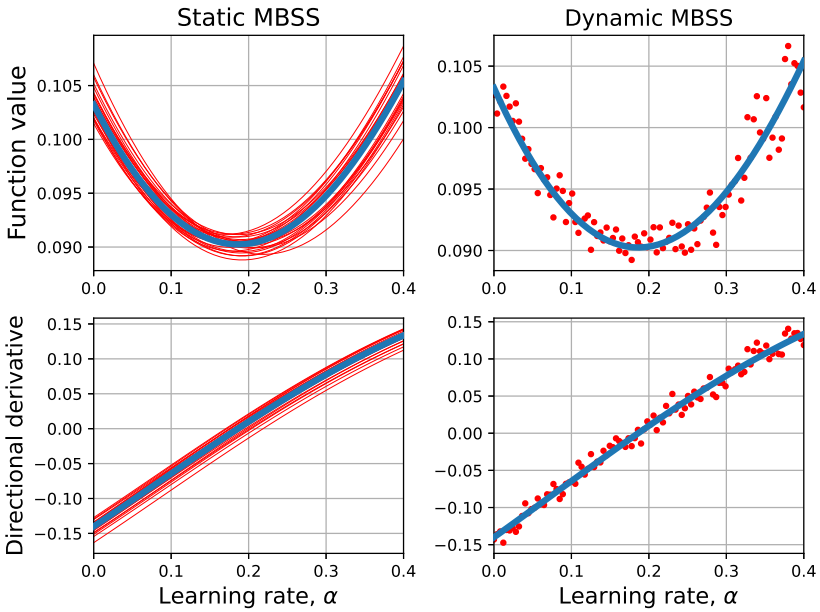


Figure 2.1: Demonstration of plotting function value and directional derivative functions for static and dynamic MBSS in red and full-batch function value and directional derivative functions in blue. A 3-hidden-layer feedforward network is used for the MNIST dataset.

uations along a search direction. Figure 2.1 illustrates function value and directional derivative plots using static and dynamic MBSS in red, with the full-batch plots in blue, using 3-hidden-layer feedforward deep neural network (DNN) [Mahsereci and Hennig, 2017] for the Modified National Institute of Standards and Technology (MNIST) dataset [LeCun et al., 1998]. Resolved minimizers on static MBSS loss functions have low variance, but high bias between static MBSS loss functions as compared to the low bias and higher variance of dynamic MBSS loss functions. The higher variance in dynamic MBSS loss functions is due to changes in sampling errors resulting from frequently updating the mini-batches [Kafka and Wilke, 2021]. The advantage of dynamic MBSS over static MBSS is that the training data throughput is significantly larger [Bottou, 2010] for the same batch size, which motivates the direction of our study to resolve the issues associated with point-wise discontinuities when aiming to resolve learning rates using univariate approximations.

MBSS has enabled the training of numerous machine learning problems, which could not be trained using full-batch sampling [Masters and Luschi, 2018]. MBSS, therefore, remains the industry norm for neural network training. Variance reduction techniques [Wang et al., 2013, Johnson and Zhang, 2013, Xiao and Zhang, 2014, Shang et al., 2018] aim to reduce sampling errors but are unable to eliminate them, therefore, still requiring optimization strategies that can optimize these loss functions.

It is, therefore, not surprising that numerous strategies for selecting efficient learning rates within a stochastic setting have been introduced. These include scheduling methods, such as cyclical learning rates [Smith, 2017] and cosine annealing [Loshchilov and Hutter, 2017a]. However, these often require new hyper-parameters to be selected. Secondly, we have adaptive learning rate methods such as AdaDelta [Zeiler, 2012a], Adam [Kingma and Ba, 2014], which resolve learning rates based on diagonal Hessian approximations. However, these including RMSprop [Tieleman and Hinton, 2012], generalize poorly compared to the stochastic gradient descent (SGD) for some cases [Wilson et al., 2017]. Lastly, line searches using sequential evaluations of the loss function, have proven useful in estimating optimizers when evaluating loss functions using static mini-batches [Friedlander and Schmidt, 2012, Byrd et al., 2011, 2012b, Bollapragada et al., 2018, Kungurtsev and Pevny, 2018, Bergou et al., 2018, Mutschler and Zell, 2019].

Attempts to resolve learning rates for dynamic MBSS loss functions are limited to two studies, namely, a probabilistic line search (PLS) [Mahsereci and Hennig, 2017] and an inexact gradient-only line search (GOLS-I) [Kafka and Wilke, 2021]. PLS's uses both function value and directional derivative information to construct a Gaussian process surrogate [Mahsereci and Hennig, 2017]. Conversely, GOLS-I conducts direct optimization along a search direction by locating stochastic non-negative gradient projection points (SNN-GPPs). SNN-GPPs manifest as a sign change, from negative to positive, in the directional derivative when moving along a descent direction [Kafka and Wilke, 2021]. SNN-GPPs are characterized as points in the design domain from which all directional derivatives around this point have a non-zero probability of being positive. GOLS-I proved to be competitive against PLS, although it uses significantly less information [Kafka and Wilke, 2021]. This raises the question of the usefulness of function value and directional derivative information when constructing approximations for dynamic MBSS loss functions, which is the subsequent focus of this study.

We investigate the quality of function value and directional derivative information in the construction of quadratic line search approximations. Specifically, we construct univariate (1-dimensional) quadratic approximations in five ways using

1. Only function values (one approximation)
2. Both function values and directional derivatives (three approximations)
3. Only directional derivative information (one approximation)

The usefulness of the various pieces of information in constructing quadratic approximations of the loss function, which is the most straightforward polynomial function with one extremum, is quantified. This implies a linear approximation of the directional derivative or a linear directional-derivative-only (related to gradient-only surrogates [Snyman and Wilke, 2018]) approximation. Although quadratic approximations may seem simplistic, they are popular in practice, which includes second order and state-of-art adaptive learning rate methods such as AdaDelta and Adam. These efficiently solve highly non-linear and ill-conditioned problems when compared to first-order approaches [Bordes et al., 2009, Bottou et al., 2018].

As a simple quadratic approximation may introduce bias into the learning rate predictions, we focus on the variance of the predicted learning rates to assess the usefulness of the approximations as opposed to purely the performance of the line searches. This is also justified from our study being focused on SGD directions. The performance of SGD may be improved by overshooting the optimizers to alleviate the resulting orthogonality of consecutive steepest descent directions, that are optimally resolved, to aim for some form of conjugacy between search directions. The usefulness of function values and directional derivatives to construct approximations are assessed on 1-layer and 3-layer deep networks for MNIST dataset [LeCun et al., 1998], and ResNet-18 [He et al., 2016] for CIFAR-10 [Krizhevsky, 2009] dataset. Our primary finding is that approximation errors that have high variance perform worse than approximation errors with high bias but low variance for training a neural network problem. Our primary recommendation is that it is essential to have directional derivative information at the initial point to reduce the variance in predicted learning rates.

2.3 Related work

The neural network full-batch loss function, $\mathcal{L}(\boldsymbol{x})$, and gradient function, $\boldsymbol{\nabla}\mathcal{L}(\boldsymbol{x})$, of weights $\boldsymbol{x} \in \mathbb{R}^w$, is given by

$$\mathcal{L}(\boldsymbol{x}) := \frac{1}{M}\sum_{b=1}^{M} \ell(\boldsymbol{x}; \boldsymbol{t}_b), \tag{2.1}$$

$$\boldsymbol{\nabla}\mathcal{L}(\boldsymbol{x}) := \frac{1}{M}\sum_{b=1}^{M} \boldsymbol{\nabla}\boldsymbol{\ell}(\boldsymbol{x}; \boldsymbol{t}_b), \tag{2.2}$$

where the individual sample loss, $\ell(\boldsymbol{x};\boldsymbol{t}_b) \in \mathbb{R}$, and gradient, $\boldsymbol{\nabla}\boldsymbol{\ell}(\boldsymbol{x};\boldsymbol{t}_b) \in \mathbb{R}^w$, is computed using each training sample, $\boldsymbol{t}_b$, taken from the whole training set, $\{\boldsymbol{t}_1, ..., \boldsymbol{t}_M\}$, of the total M samples. Mini-batch sub-sampling [Robbins and Monro, 1951] loss, $L(\boldsymbol{x})$, and gradient, $\boldsymbol{g}(\boldsymbol{x})$, are given by

$$L(\boldsymbol{x}) := \frac{1}{|\mathcal{B}|}\sum_{b\in\mathcal{B}} \ell(\boldsymbol{x};\boldsymbol{t}_b), \tag{2.3}$$

$$\boldsymbol{g}(\boldsymbol{x}) := \frac{1}{|\mathcal{B}|}\sum_{b\in\mathcal{B}} \boldsymbol{\nabla}\boldsymbol{\ell}(\boldsymbol{x};\boldsymbol{t}_b), \tag{2.4}$$

where the uniform-randomly sub-sampled mini-batch, $\mathcal{B} \subset \{1, 2, ..., M\}$, of subset size, $|\mathcal{B}| \ll M$ is used, instead of the full-batch. Mini-batch sub-sampling (MBSS) can be performed either statically or dynamically, concerning line search methods, which require multiple function or gradient evaluations to be computed along every n-th search direction, $\boldsymbol{d}_n$. Static MBSS re-samples a new mini-batch for every search direction, $\boldsymbol{d}_n$. Conversely, dynamic MBSS, which this paper focuses on, resamples a new mini-batch for every function computation; hence, the mini-batch is updated multiple times along a search direction. The dynamic MBSS loss, $\tilde{L}(\boldsymbol{x})$, and gradient functions, $\tilde{\boldsymbol{g}}(\boldsymbol{x})$, are given by

$$\tilde{L}(\boldsymbol{x}) := \frac{1}{|\mathcal{B}_{n,i}|}\sum_{b\in\mathcal{B}_{n,i}} \ell(\boldsymbol{x};\boldsymbol{t}_b), \tag{2.5}$$

$$\tilde{\boldsymbol{g}}(\boldsymbol{x}) := \frac{1}{|\mathcal{B}_{n,i}|}\sum_{b\in\mathcal{B}_{n,i}} \boldsymbol{\nabla}\boldsymbol{\ell}(\boldsymbol{x};\boldsymbol{t}_b), \tag{2.6}$$

where $\mathcal{B}_{n,i}$ denotes the dynamically sub-sampled mini-batch at the i-th function evaluation for the n-th search direction.

Unlike static MBSS for which the loss function is smooth and continuous, dynamic MBSS loss functions are point-wise discontinuous or stochastic resulting in many local minima when using the minimization approaches. The n-th line search is given by,

$$\alpha_n^* = \arg\min_{\alpha_n} \tilde{L}(\boldsymbol{x}_n + \alpha_n \boldsymbol{d}_n), \tag{2.7}$$

where α_n^* and α_n denote the optimal learning rate and learning rate at the n-th iteration, respectively.

To solve the issues resulting from point-wise discontinuities, a Gaussian process surrogate model based on Bayesian optimization was proposed by Mahsereci and Hennig [2017] to estimate the expected local minima by satisfying the Wolfe condition [Wolfe, 1969, 1971] in a probabilistic sense for stochastic gradient descent, and, furthermore, Wills and Schön [2018] proposed a probabilistic quasi-Newton method. Kafka and Wilke [2021] developed an inexact gradient-only line search (GOLS-I), that explicitly evaluates directional derivatives to locate sign changes from negative to positive along descent directions.

GOLS-I is based on the principles underlying gradient-only optimization [Snyman and Wilke, 2018], which aim to resolve Stochastic Non-Negative Associated Gradient Projection Points (SNN-GPPs), $\boldsymbol{x}_{snngpp}$, which satisfy

$$p(\boldsymbol{d}^\top \cdot \tilde{\boldsymbol{g}}(\boldsymbol{x}_{snngpp} + \epsilon\boldsymbol{d}) \geq 0) > 0, \quad \forall \|\boldsymbol{d} \in \mathbb{R}^w\|_2 = 1, \quad \forall \epsilon \in (0, \epsilon_{max}], \tag{2.8}$$

where $p(*)$ denotes the probability [Kafka and Wilke, 2021]. This states that there is always a learning rate, travelled along $\boldsymbol{d}$ from $\boldsymbol{x}_{snngpp}$, which results in $p > 0$ for having a positive directional derivative. The proposed directional-derivative-only approximations aim to resolve $\boldsymbol{x}_{snngpp}$, since no function value information is used to construct the approximation. In turn, the zero-order approximation aims to resolve minimizers (2.7) as it only uses function value information. The other three approximations use both function value and directional derivative information.

2.3.1 Using approximations for line searches

The two main reasons for investigating univariate approximations for line searches to training DNNs are i) computational efficiency [Vu et al., 2017, Bhosekar and Ierapetritou, 2018] and ii) higher-dimensional surrogates are susceptible to the "curse of dimensionality" [Kubicek et al., 2015].

Approximation (or surrogate) models are often used to analyze and optimize expensive black-box problems [Vu et al., 2017, Bhosekar and Ierapetritou, 2018]. They are usually constructed by regressing through function values sampled in the domain of interest because gradients are often not available for many applications [Regis and Shoemaker, 2013]. However, when gradients are available, they are likely to improve the quality of the models [Chen et al., 2019, Laurent et al., 2019]. In neural network training, analytical gradient information is readily available through back-propagation [Rumelhart et al., 1986].

Higher-dimensional approximations (or surrogates) are susceptible to the "curse of dimensionality" [Kubicek et al., 2015], making them unsuited for high-dimensional problems like DNNs. However, univariate line searches resolve this issue and require fewer data points as compared to higher-order approximations. Furthermore, Wilke [2016], Snyman and Wilke [2018] showed that continuous approximations could also be constructed using only directional derivative information. In this study, we quantify the effect of enforcing different information, such as function value and directional derivatives, at different spatial locations along a descent direction in the construction of univariate approximations.

Approximation errors may appear as biases and variances. Let us consider two extreme cases of biases and variances to show the implications of them to line searches. When approximation errors have large bias but no variance, we know that the approximation errors are deterministic and improving the approximation model would help reduce the errors. However, when approximation errors have no bias but large variance, the approximation errors are stochastic and therefore, also the learning rates. To reduce the variance of the approximation predicted learning rates, one can choose information used to construct the approximation that reduces the variance. Hence, for a problem which produces noisy function value and directional derivative information, we prefer to find an approximation with a larger bias than a larger variance.

Using approximations for line searches for training DNNs

Research over the last few years proved promising when using line search approximations to resolve learning rates during DNNs training. Gaussian process approach based on Bayesian optimization sampled using dynamic MBSS to implement a univariate (1-D) line search [Mahsereci and Hennig, 2017], and recently, univariate parabolic approximations were introduced to resolve learning rates for static MBSS loss functions [Mutschler and Zell, 2019]. Both approaches use function values and directional derivative information to construct approximations.

In the next section, we propose five parabolic approximations, constructed using different sets of information, to study the characteristics and effects of using function values and directional derivatives computed at various locations for training DNNs.

2.4 Enforcing selective information to construct 1D quadratic approximations

In the construction of 1D quadratic approximations, selected information is enforced at different points along the descent direction. Specifically, we consider the following five approximations using function value and/or directional derivative information at $\alpha_0 = 0$, $\alpha_1 > 0$ and $\alpha_2 > 0$, with $\alpha_1 > \alpha_2$:

1. f-f-f (function-value-only): the function values at α_0, α_1 and α_2, respectively, denoted by $\tilde{f}_0$, $\tilde{f}_1$ and $\tilde{f}_2$;

2. fg-f (mixed): the function value and directional derivative at α_0 and the function value at α_1, respectively, denoted by $\tilde{f}_0$, $\tilde{f}'_0$ and $\tilde{f}_1$;

3. f-fg (mixed): the function value at α_0 and the function value and directional derivative at α_1, respectively, denoted by $\tilde{f}_0$, $\tilde{f}_1$ and $\tilde{f}'_1$;

4. fg-fg (mixed): the function value and directional derivative at α_0 and the function value and directional derivative at α_1, respectively, denoted by $\tilde{f}_0$, $\tilde{f}'_0$, $\tilde{f}_1$ and $\tilde{f}'_1$;

5. g-g (derivative-only): the directional derivatives measured at both α_0 and α_1, respectively, denoted by $\tilde{f}'_0$ and $\tilde{f}'_1$.

Note that we do not consider fg-g and g-fg, which use one function value either at α_1 or α_2, respectively and directional derivatives at both points, because they produce the identical local minima along α as g-g.

A multivariate dynamic MBSS loss function, $\tilde{\mathcal{L}} : (\boldsymbol{x} \in \mathbb{R}^w) \to \mathbb{R}$, along the search direction, $\boldsymbol{d}$, is given by the univariate function

$$\tilde{f}(\alpha) := \tilde{\mathcal{L}}(\boldsymbol{x} + \alpha \boldsymbol{d}). \tag{2.9}$$

The 1D quadratic approximation function, $\hat{f}(\alpha)$, of $\tilde{f}(\alpha)$ is denoted by

$$\hat{f}(\alpha) = k_1\alpha^2 + k_2\alpha + k_3 \approx \tilde{f}(\alpha). \tag{2.10}$$

Similarly, the directional derivative function along $\boldsymbol{d}_n$ is denoted by

$$\tilde{f}'(\alpha) = \boldsymbol{d}^{\mathrm{T}}\tilde{\boldsymbol{g}}(\boldsymbol{x} + \alpha \boldsymbol{d}) \tag{2.11}$$

and this is approximated by the following linear gradient approximation function:

$$\hat{f}'(\alpha) = 2k_1\alpha + k_2 \approx \tilde{f}'(\alpha). \tag{2.12}$$

The approximation $\tilde{f}$ required at least three pieces of independent information to resolve k_1, k_2 and k_3, while $f'(\alpha)$ requires only two pieces of independent information to resolve k_1 and k_2. The function-value-only and mixed models are constructed using the quadratic approximation function (2.10), and the derivative-only model is constructed using a linear approximation function (2.12). The five approximations with the required information (red circle indicates respective function value and red slope indicates the derivative of the respective function) to construct each is shown in Figure 2.2. All approximations are well-specified linear systems except for the over-specified mixed (fg-fg) approximation, that requires a least-squares solution to approximate four pieces of information using only three variables.

Each approximation can be constructed by solving a small linear system, $\boldsymbol{A}_i\boldsymbol{k}_i = \boldsymbol{b}_i$, $i = 1, 2, 3, 4, 5$, that is obtained by evaluating (2.10) and/or (2.12) at selected α values to enforce/approximate the respective function or derivative information, with $\boldsymbol{A}_i$ and $\boldsymbol{b}_i$ respectively given by

$$\boldsymbol{A}_1 = \begin{bmatrix} 0 & 0 & 1 \\ \alpha_1^2 & \alpha_1 & 1 \\ \alpha_2^2 & \alpha_2 & 1 \end{bmatrix}; \boldsymbol{b}_1 = \begin{bmatrix} \tilde{f}_0 & \tilde{f}_1 & \tilde{f}_2 \end{bmatrix}^T \tag{2.13}$$

$$\boldsymbol{A}_2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ \alpha_1^2 & \alpha_1 & 1 \end{bmatrix}; \boldsymbol{b}_2 = \begin{bmatrix} \tilde{f}_0 & \tilde{f}'_0 & \tilde{f}_1 \end{bmatrix}^T \tag{2.14}$$

$$\boldsymbol{A}_3 = \begin{bmatrix} 0 & 0 & 1 \\ \alpha_1^2 & \alpha_1 & 1 \\ 2\alpha_1 & 1 & 0 \end{bmatrix}; \boldsymbol{b}_3 = \begin{bmatrix} \tilde{f}_0 & \tilde{f}_1 & \tilde{f}'_1 \end{bmatrix}^T \tag{2.15}$$

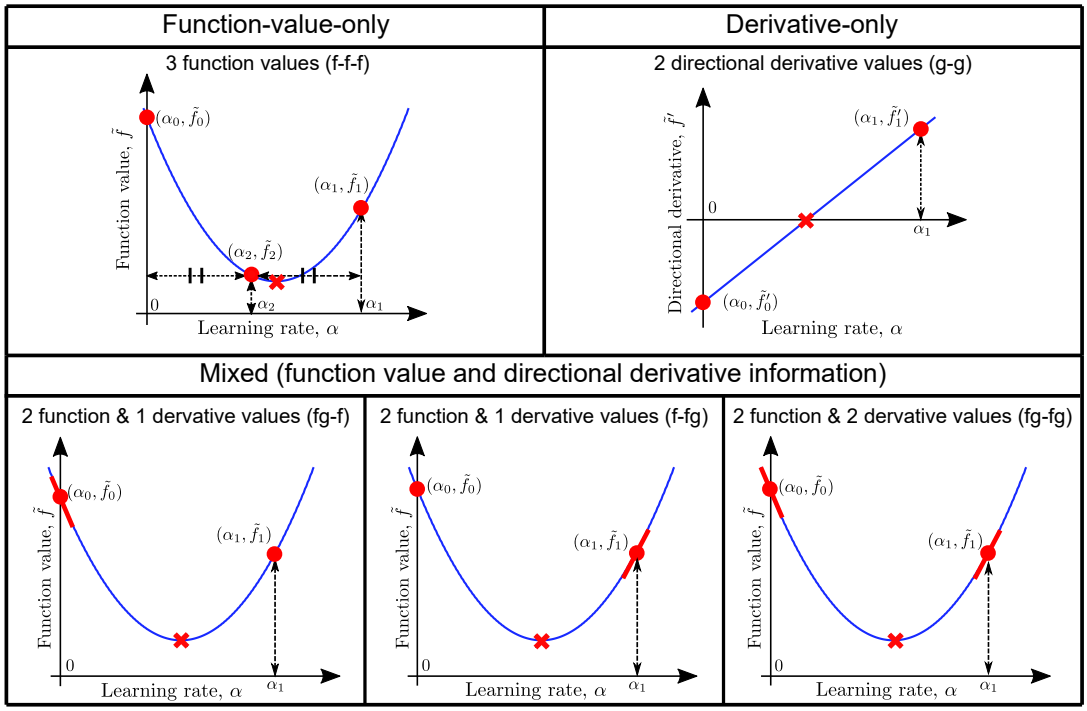


Figure 2.2: Illustration of quadratic approximations using different types of information; (top left) the function-value-only approximation (f-f-f), (bottom right) the mixed approximation with the directional derivative at α_0 (fg-f), (bottom middle) the mixed approximation with the directional derivative at α_1 (f-fg), (bottom right) the mixed approximation with the directional derivatives at both α_0 and α_1 (fg-fg) and (top right) derivative-only approximation (g-g).

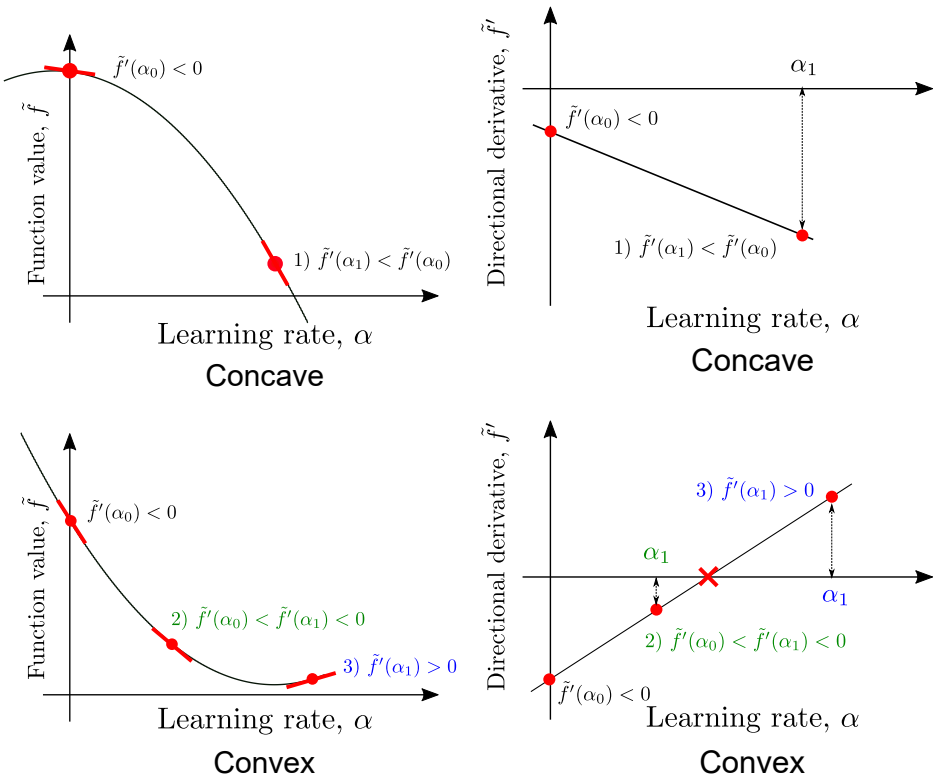
$$\boldsymbol{A}_4 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ \alpha_1^2 & \alpha_1 & 1 \\ 2\alpha_1 & 1 & 0 \end{bmatrix}; \boldsymbol{b}_4 = \begin{bmatrix} \tilde{f}_0 & \tilde{f}'_0 & \tilde{f}_1 & \tilde{f}'_1 \end{bmatrix}^T \tag{2.16}$$

$$\boldsymbol{A}_5 = \begin{bmatrix} 0 & 1 \\ 2\alpha_1 & 1 \end{bmatrix}; \boldsymbol{b}_5 = \begin{bmatrix} \tilde{f}'_0 & \tilde{f}'_1 \end{bmatrix}^T \tag{2.17}$$

Because we use the linear and quadratic approximation models to predict the optimal learning rate, α_n^*, approximation error is inevitable, and it may be in forms of either undershooting or overshooting. There are three possible scenarios when computing α_n^* from a 1D quadratic approximation (2.10) or linear derivative approximation (2.12): i) unbounded extrapolation, ii) bounded extrapolation and iii) interpolation, depending on whether the approximation is convex, $k_1 > 0$, or concave, $k_1 < 0$, as shown in Figures 2.3(a) and (b) for the function value and derivative approximations, respectively. The three possible extrapolation or interpolation scenarios are as follows:

1. If $\tilde{f}'_1 < \tilde{f}'_0 < 0$, the approximation becomes concave for unbounded extrapolation;
2. If $\tilde{f}'_0 < \tilde{f}'_1 < 0$, the approximation becomes convex for bounded extrapolation, with local minimum α^* at $\alpha^* > \alpha_1$;
3. If $\tilde{f}'_1 > 0$, the approximation interpolates, and there is a local minimum α^* at $\alpha_0 < \alpha^* < \alpha_1$.

Both unbound and bound extrapolations predict the local minima, α_n^*, beyond the largest observation point, α_1. This may cause a large prediction error when α_n^* is far from α_1. Hence, we only allow for the interpolation scenario when computing α_n^* in the study.



(a) Function value domain (b) Directional derivative domain

Figure 2.3: Illustration of 1) unbounded extrapolation and 2) bounded extrapolation 3) interpolation situations for approximations in (a) function value or (b) directional derivative domain.

2.5 Pseudocode and implementation details

The general pseudocode is presented in Algorithm 1. The line search algorithm starts with the inputs of the current mini-batch gradient $\tilde{\boldsymbol{g}}_{0,n}$ and search direction $\boldsymbol{d}_n$. The direction $\boldsymbol{d}$ only needs to be a valid descent direction, but in this paper, we use the SGD directions for simplicity, $\boldsymbol{d}_n = -\tilde{\boldsymbol{g}}_{0,n}$ at α_0.

Algorithm 1: Line search approximation

Input: $\tilde{\boldsymbol{g}}_{0,n} = \tilde{\boldsymbol{g}}^*_{n-1}$, $\boldsymbol{d}_n$, $\tilde{f}_{0,n} = \tilde{f}^*_{n-1}$, ε, α_{min}, α_{max}
Output: α^*_n, $\tilde{\boldsymbol{g}}^*_n$, $\tilde{f}^*_n$

Compute directional derivative $\tilde{f}'_{0,n}$ at α_0
if $|\tilde{f}'_{0,n}| < \varepsilon$ **then**
  Recompute $\tilde{\boldsymbol{g}}_{0,n}$ at α_0 with a new sub-sampled mini-batch for the next iteration.
  return $\alpha^*_n = 0$, $\tilde{\boldsymbol{g}}^*_n$, $\tilde{f}^*_n$
$\alpha_1 = \max(\alpha_{min}, \min(\|\boldsymbol{d}_n\|_2^{-1}, \alpha_{max}))$
`/* Option 1: Function-value-only approximation (f-f-f) */`
Compute $\tilde{f}_{1,n}$ at α_1 and $\tilde{f}_{2,n}$ at $\alpha_2 = \alpha_1/2$
$\alpha^* =$ `StepSizeFFF`$(\alpha_1, \alpha_2, \tilde{f}_{0,n}, \tilde{f}_{1,n}, \tilde{f}_{2,n})$; `// Algorithm 2`
`/* Option 2: Mixed approximation (fg-f) */`
Compute $\tilde{f}_{1,n}$ at α_1
$\alpha^* =$ `StepSizeFGF`$(\alpha_1, \tilde{f}_{0,n}, \tilde{f}_{1,n}, \tilde{f}'_{0,n})$; `// Algorithm 6 in Appendix A.1`
`/* Option 3: Mixed approximation (f-fg) */`
Compute $\tilde{f}_{1,n}, \tilde{f}'_{1,n}$ at α_1
$\alpha^* =$ `StepSizeFFG`$(\alpha_1, \tilde{f}_{0,n}, \tilde{f}_{1,n}, \tilde{f}'_{1,n})$; `// Algorithm 7 in Appendix A.1`
`/* Option 4: Mixed approximation (fg-fg) */`
Compute $\tilde{f}_{1,n}, \tilde{f}'_{1,n}$ at α_1
$\alpha^* =$ `StepSizeFGFG`$(\alpha_1, \tilde{f}_{0,n}, \tilde{f}_{1,n}, \tilde{f}'_{0,n}, \tilde{f}'_{1,n})$; `// Algorithm 8 in Appendix A.1`
`/* Option 5: Derivative-only approximation (g-g) */`
Compute $\tilde{f}'_{1,n}$ at α_1
$\alpha^* =$ `StepSizeGG`$(\alpha_1, \alpha_2, \tilde{f}'_{0,n}, \tilde{f}'_{1,n})$; `// Algorithm 9 in Appendix A.1`
if $(\alpha^* \neq \alpha_1)$ & $(0 < \alpha^* < \alpha_1)$ **then**
  $\tilde{\boldsymbol{g}}^*_n = \tilde{\boldsymbol{g}}_n(\boldsymbol{x}_0 + \alpha^* \boldsymbol{d}_n)$
else
  $\alpha^*_n = \alpha_1$
return α^*_n, $\tilde{\boldsymbol{g}}^*_n$, $\tilde{f}^*_n$

In Algorithm 1, we first check whether the magnitude of the directional derivative $|\tilde{f}'_{0,n}|$ is greater than a prescribed tolerance of $\varepsilon \approx 10^{-16}$. The magnitude needs to be large enough not to cause any numerical errors. Otherwise, the local stochastic gradient, $\tilde{\boldsymbol{g}}_{0,n}$, is recomputed using a new randomly sampled mini-batch to be returned for the next iteration with the final learning rate of $\alpha^*_n = 0$. For sufficiently large $|\tilde{f}'_{0,n}|$, an initial learning rate, α_1, based on the magnitude of the direction vector, $\boldsymbol{d}_n$, is chosen. We select it to be the inverse of the l_2-norm of the search direction, $\|\boldsymbol{d}_n\|_2^{-1}$, as this provides small and exploiting learning rates for steeper directional derivatives, $|f'_0| \gg 0$, and large and exploring learning rates for flatter directional derivatives, $|f'_0| \approx 0$. We still make sure that α_1 falls within the prescribed bounds denoted by α_{min} and α_{max}, which are bounded to 10^{-7} and 10^8 in this study, as used by Kafka and Wilke [2021]. This extremely large choice of learning rate may demonstrate the robustness of the algorithm.

Once, the initial guess, α_1, is finalized, we can compute relevant information required to construct the approximation of our choice. Since we already have $\tilde{f}_{0,n}$ and $\tilde{f}'_{0,n}$ available, we need to compute additionally

1. $\tilde{f}_{1,n}$ and $\tilde{f}_{2,n}$ for the f-f-f approximation,
2. $\tilde{f}_{1,n}$ for the fg-f approximation,
3. $\tilde{f}_{1,n}$ and $\tilde{f}'_{1,n}$ for the f-fg and fg-fg approximations,
4. $\tilde{f}'_{1,n}$ for the g-g approximation.

	Function evaluations	
Approx.	Resample	Approximation
f-f-f	3	5
fg-f	3	4
f-fg	3	6
fg-fg	3	6
g-g	3	6

Table 2.1: Required number of function evaluations for each approximation for resampling and building an approximation.

The learning rates for each approximation are computed using `StepSizeFFF` (Algorithm 2), `StepSizeFGF`, `StepSizeFFG`, `StepSizeFGFG` and `StepSizeGG` (Algorithms 6-9 in Appendix A.1) for the f-f-f, fg-f, f-fg, fg-fg and g-g approximations, respectively.

Algorithm 2: `StepSizeFFF`

Input: α_1, α_2, $\tilde{f}_0$, $\tilde{f}_1$, $\tilde{f}_2$, ε
Output: α^*
1 $\alpha^* = \alpha_1$
2 Define a matrix $\boldsymbol{A}_1$ and a vector $\boldsymbol{b}_1$ from (2.13)
3 **if** $rank(\boldsymbol{A}_1) = 3$ **then**
4 Solve for the constants $\boldsymbol{k} = \boldsymbol{A}_1^{-1}\boldsymbol{b}_1$ from (2.10)
5 **if** $k_1 > \varepsilon$ **then**
6 $\alpha^* = \max(\alpha_{min}, \min(-k_2/(2k_1), \alpha_{max}))$

The respective learning rate functions ensure that the matrix $\boldsymbol{A}$ is not singular (e.g. line 3 in Algorithm 2). We check whether the curvature k_1 is a positive value or larger than a prescribed tolerance ε, which we set to be 10^{-16}. This ensures that the approximation is convex and that it does not run into numerical issues for computing the learning rate α^* (e.g. line 5 in Algorithm 2).

Again, we check if the computed learning rate is within the upper and lower bounds (α_{max} and α_{min}). Note that the linear system of the fg-fg approximation (2.16) is solved using least-squares regression or Moore-Penrose pseudoinverse and that when implementing the g-g case, the rank of the matrix $\boldsymbol{A}_5$ (2.17) needs to be 2, since there are only 2 constants (i.e. k_1 and k_2) that are solved.

Table 2.1 shows that the required number of function evaluations may vary between approximation types, and conditions enforced in an iteration. The different conditions are

1. Resample when the initial magnitude of the directional derivative is too small (line 2 in Algorithm 1);

2. Constructing approximations for computing α^*.

We count the number of function evaluations required for forward pass and backward pass as 1 and 2, respectively. Note, the gradient at a point can only be computed once the forward pass has been conducted. Therefore, g-g requires 6 function evaluations for interpolation in total. Additionally, we always require the gradient at the starting point, α_0, for the search direction, whether it is used or not for constructing the model, f-f-f, for example, needs 5 function evaluations. Hence, the fg-f approximation is computationally inexpensive as it requires only 4 function evaluations.

2.6 Experimental setup

In this section, we consider numerical investigations to examine the performance and characteristics of the five approximation models in Algorithm 1. First, we take the experimental study as outlined by Mahsereci and Hennig [2017] for the MNIST dataset [LeCun et al., 1998], which

is a 10-class classification dataset with 28×28 input size, 6×10^4 train samples and 1×10^4 test samples. The batch sizes chosen for MNIST are 10, 100, 200 and 1000. The maximum number of function evaluations is 4×10^4. The MNIST dataset is trained on two fully connected feedforward nets with biases: N-I and N-II.

- N-I: it has a shallow network structure of a single hidden layer; $\mathcal{N}_{input}$-800-$\mathcal{N}_{ouput}$ with sigmoid activation functions and a cross-entropy loss. The initial weights are sampled from a normal distribution.
- N-II: it has a deep network structure of three hidden layers; $\mathcal{N}_{input}$-1000-500-250-$\mathcal{N}_{output}$ with tanh activation functions and a square loss function. The initial weights are obtained using the Xavier initialization [Glorot and Bengio, 2010].

Note that all the problems are one-hot encoded. The training for each approximation is conducted ten times at different starting points to compute the mean and standard deviation. For this experiment, we compare the performance of the five approximation models against 1) Adam using the default PyTorch settings [Paszke et al., 2019] and 2) scheduled SGD (SSGD) [Liu, 2020] with diminishing steps size by a factor of 10 at 3/7 and 5/7 of the maximum number of function evaluations, starting from the initial learning rate of 0.1.

Secondly, we also conduct investigations using ResNet-18 [He et al., 2016] on the CIFAR-10 dataset. CIFAR-10 is a 10-class classification dataset with 32×32 input size, 5×10^4 training samples and 1×10^4 testing samples. The results are averaged over five runs using a chosen batch size of 128 [He et al., 2016]. The maximum number of epochs is 350. We based our ResNet-18 PyTorch implementation on the source code provided by Liu [2020] to implement the ResNet-18 problem. We compare the results against SSGD with diminishing steps size by a factor of 10 at 150 and 250 epochs, starting from the initial learning rate of 0.1 as proposed by Liu [2020] except we omitted weight decay and momentum for fair learning rate comparisons.

Note that the search direction for the five approximations in both experiments is set to be the stochastic gradient descent (SGD) direction. The minimum and the maximum learning rates are chosen for the five approximations are $\alpha_{min} = 10^{-8}$ and $\alpha_{max} = 10^{7}$, respectively. The wide limit ranges allow us to investigate the robustness of the various approximations. Also, note that both SSGD and Adam are state-of-art optimizers that include momentum. The implications are that their search directions are distinct from gradient descent that is used for the five approximations. Hence, the performances of SSGD and Adam are a combination of their learning rates as well as their descent directions. The reader is, therefore, cautioned not to overinterpret the comparisons to the five approximations using gradient decent directions.

2.7 Experimental results

The results for N-I and N-II for the MNIST dataset are shown in Figures 2.4 and 2.5, respectively. We compare the performance of the five approximation models with Adam and SSGD. For each figure, from the top row, each row shows train errors, test errors, and learning rates in $\log_{10}$ scale, respectively, and the batch size, $|\mathcal{B}|$, increases from left to right.

From Figure 2.4, it is evident the initial training errors are high for all methods when the smallest batch size, $|\mathcal{B}| = 10$, is considered. This is due to the high variance in the computed gradients. The five approximations show significant improvements, as $|\mathcal{B}|$ increases, outperforming the SSGD. Adam, which its search direction is based on the estimates of 1st and 2nd moments of the gradients, shows the best performance. The test errors show only slight improvements, as $|\mathcal{B}|$ increases for all methods. The five approximations show that their learning rates grow, as training continues. This is due to the quadratic approximation curvature reducing as the problems converge. Note that we do not show the learning rates of Adam.

In Figure 2.5, we show the results for the deeper net, N-II. As $|\mathcal{B}|$ increases, the training errors generally decrease, except for the following approximations: f-f-f, f-fg, and fg-fg. It also shows that g-g and fg-f approximations result in lower train errors than Adam for smaller $|\mathcal{B}|$,

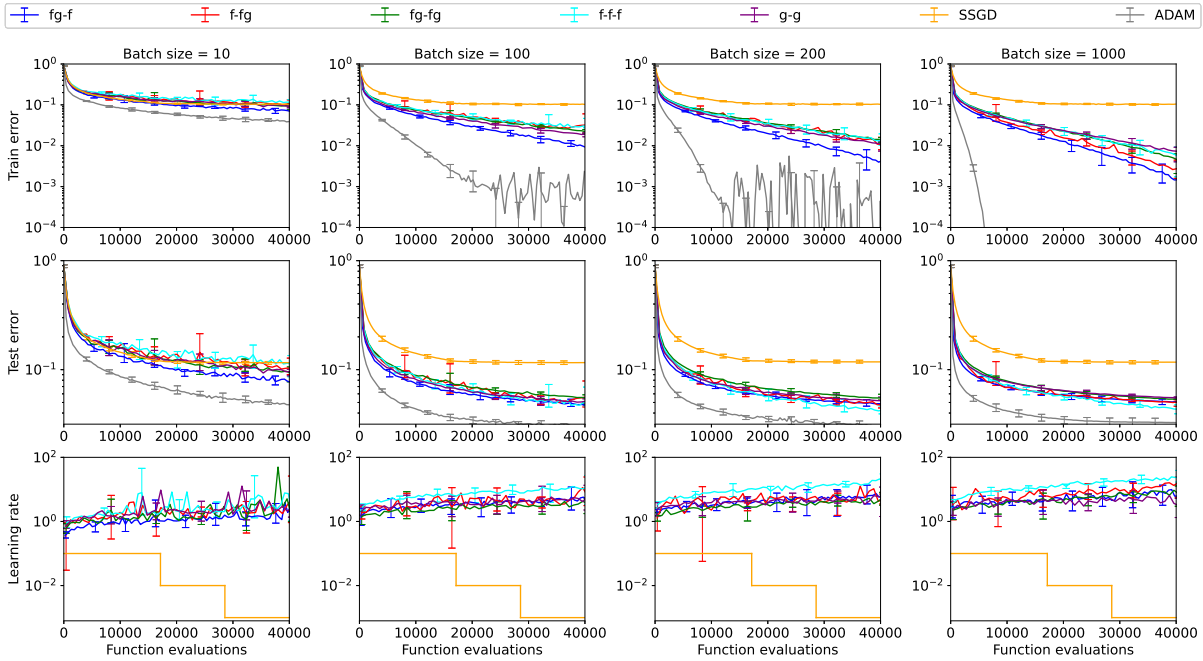


Figure 2.4: Comparison of the required number of function evaluations of the five approximation-assisted line search methods: 1) directional-derivative-only (g-g), 2) function-value-only (f-f-f), 3) mixed approximation with function values at both α_0 and α_1 and directional derivative only at α_0 (fg-f), 4) directional derivative only at α_1 (f-fg) and 5) directional derivatives at both α_0 and α_1 (fg-fg), as well as ADAM and SSGD on MNIST dataset with N-I architecture.

although ADAM converges initially faster. The similar trend also appears in the test errors. The variance in the SSGD results is lower than the other methods, but it only outperforms the poor performing approximations that include f-f-f, f-fg, and fg-fg.

This clearly indicates that the type of information and the spatial location thereof need to be carefully considered. Above all, it indicates that more information is not necessarily better as is evident with the g-g approximation outperforming the f-f-f and fg-fg approximations. The results show that either not enforcing directional derivative information at α_0, or enforcing too much information, may result in large approximation errors, especially, when using larger $|\mathcal{B}|$ for the deeper network, N-II, as opposed to smaller $|\mathcal{B}|$ for the shallow network, N-I. The learning rates for f-f-f, f-fg, and fg-fg approximations are significantly smaller than the learning rates resulting from g-g, fg-f and SSGD.

To motivate this statement further, we also investigated the characteristics of the five approximations for the N-I and N-II architectures, in Figures 2.6 and 2.7, respectively, by plotting the variance of the model minimum, α^*, resulted from each approximation, and quantify the characteristics of the solutions. It shows the distributions of the optimizers for the five approximations by resampling the mini-batch 50 times, starting at the initial random weights. Each approximation is constructed using dynamic MBSS with $|\mathcal{B}| = 10$. For direct comparison, all approximations are depicted in the loss domain and directional derivative domain in Figure 2.6(a) and (b), respectively, for N-I. In Figure 2.6(a), we converted the directional-derivative-only approximation (g-g) to the loss function domain, integrating the approximation w.r.t learning rate α and choosing an arbitrary value for the integration constant.

Note that, in Figure 2.6, the approximation error appears as large variance to the ones that do not use directional derivative information at the origin: f-f-f and f-fg, compared to the ones use the information: fg-f, fg-fg, g-g. Recall we did not consider fg-g and g-fg cases, since it would produce the identical learning rate results as g-g. Hence, this implies that using directional derivative information is essential for reducing the variance of the approximation error and consequently predicted learning rates.

The approximation error for N-II, shown in Figure 2.7, appears similar across the different

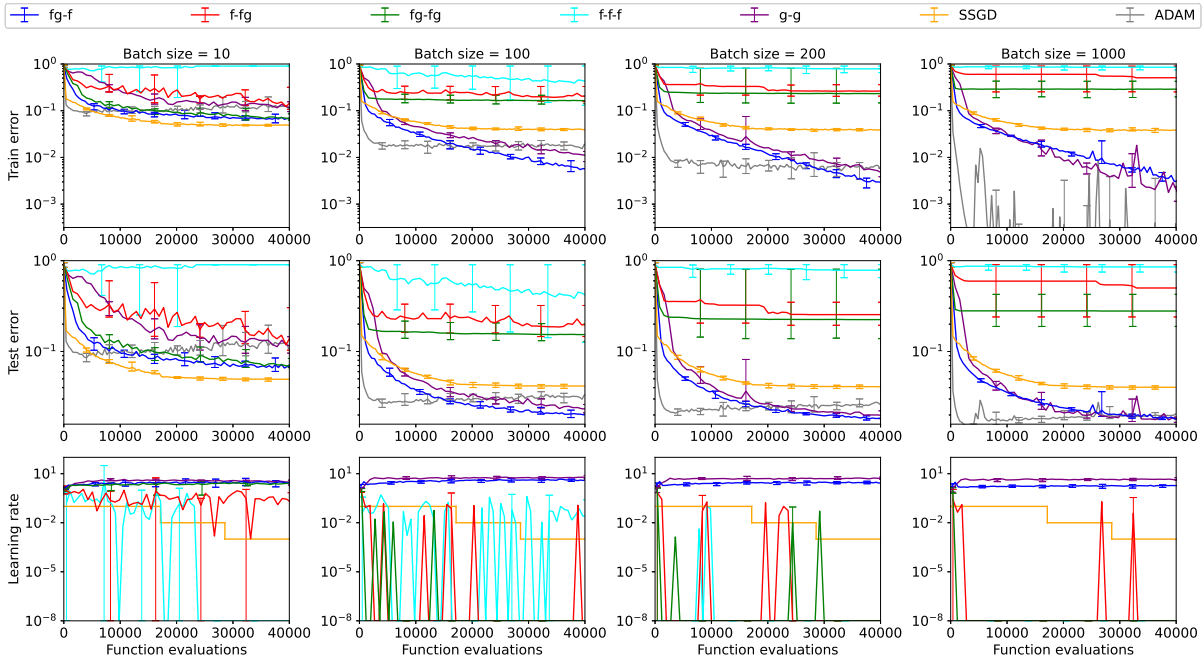
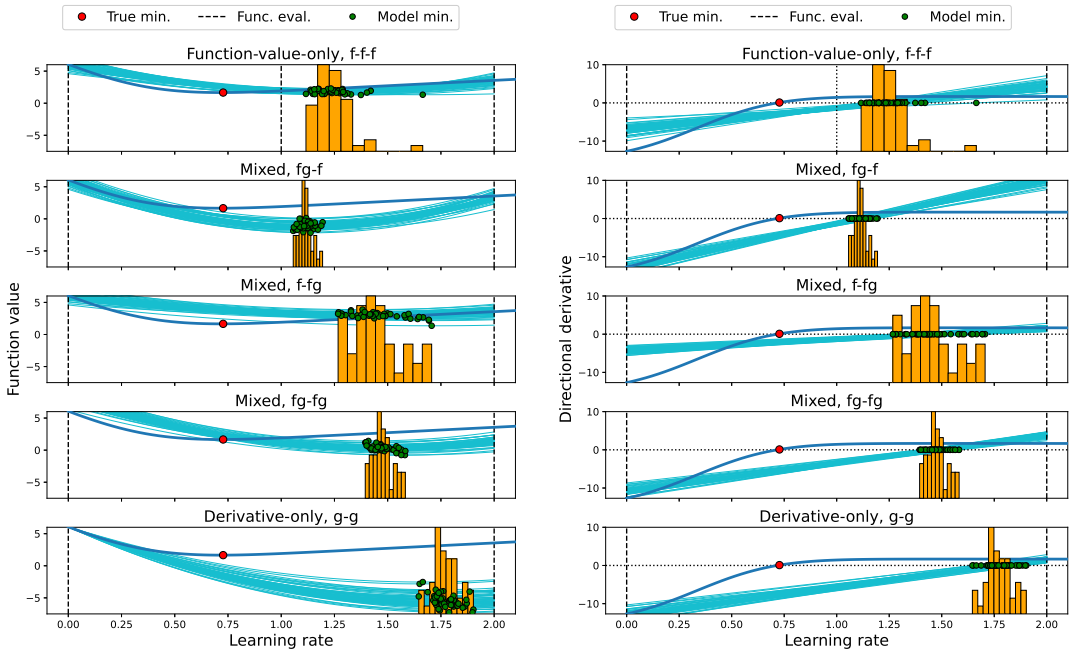


Figure 2.5: Comparison of the required number of function evaluations of the five approximation-assisted line search methods: 1) directional-derivative-only (g-g), 2) function-value-only (f-f-f), 3) mixed approximation with function values at both α_0 and α_1 and directional derivative only at α_0 (fg-f), 4) directional derivative only at α_1 (f-fg) and 5) directional derivatives at both α_0 and α_1 (fg-fg), as well as ADAM and scheduled SSGD on MNIST dataset with N-II architecture.

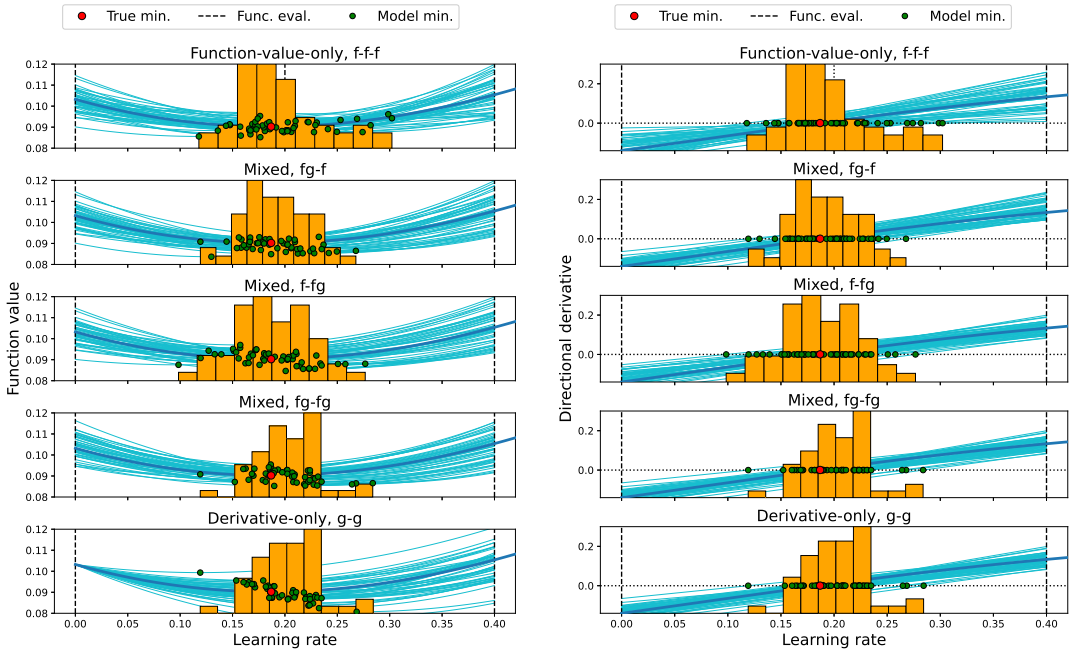
types of approximations. This is due to the closer distance to the first local minima from the initial guess for N-II, which lowers the quadratic approximation error. However, as we observed from the train and test errors in Figure 2.5, f-f-f, f-fg and fg-fg performs poorly during the training for N-II. Recall that f-f-f and fg-f are the two approximations which showed large variance in the approximation errors in Figure 2.6, since approximation errors with high variance are worse than high bias as discussed in Section 2.3.1. We also learn that fg-fg that regresses through an excessive amount of noisy data may lead to unreliable training results, as shown in Figure 2.4.

Let us consider the results of ResNet-18 for the CIFAR-10 dataset shown in Figure 2.8, and the best results are recorded in Table 2.2, which shows that g-g, fg-f, fg-fg and SSGD significantly outperform f-f-f and f-fg for both train and test accuracies. Although the SSGD was implemented by Liu [2020] to obtain high accuracy for precisely this problem, it did not perform the best among the other methods when optimizing N-I and N-II. In Figure 2.8, from left to right, it shows train accuracy, test accuracy in percentage, and the learning rates in log_{10} scale over 350 epochs. Note that fg-f, g-g and fg-fg perform similarly when observing the training and testing accuracy, while f-fg and f-f-f underperform. As before, enforcing directional derivative information at the origin is essential. Previously, fg-fg was considered to enforce too much poor quality information, which in this case performs much better. In the regression enforcement, it could be that the directional derivative magnitudes are now much larger relative to the function values than for the previous problems. This implies that directional derivative information is enforced stronger than the function values. SSGD performs the best and only marginally so, but required extensive tuning to do so [Liu, 2020]. It is interesting to note that the performance difference between SSGD and the g-g, fg-f, and fg-fg approximations is marginal, but the learning rates are distinctly different. learning rates for SSGD decrease over epochs, while g-g, fg-f and fg-fg approximations it increases quite aggressively. The studies show that both g-g and fg-f show potential to be further developed as line search algorithms.



(a) Function value (b) Directional derivative

Figure 2.6: N-I minimizer variance for the five approximations by resampling the mini-batches 50 times: f-f-f, fg-f, f-fg, fg-fg, and g-g. The derivative-only approximations are plotted as a loss function by adding arbitrary constants.



(a) Function value (b) Directional derivative

Figure 2.7: N-II minimizer variance for the five approximations by resampling the mini-batches 50 times: f-f-f, fg-f, f-fg, fg-fg, and g-g. The derivative-only approximations are plotted as a loss function by adding arbitrary constants.

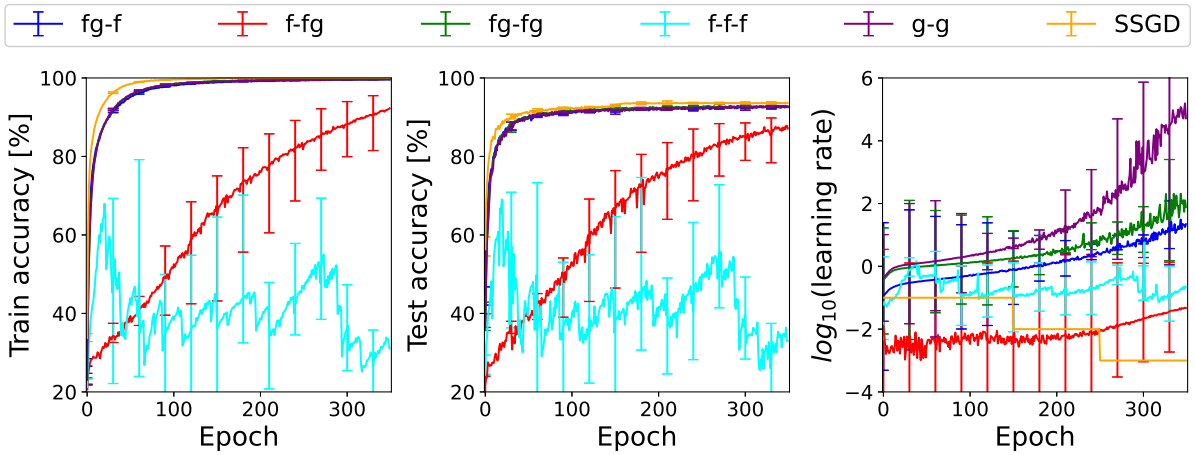


Figure 2.8: Comparison on ResNet-18 for the CIFAR-10 dataset for the following approximations: f-f-f, fg-f, f-fg, fg-fg, g-g and SSGD.

	g-g	f-f-f	fg-f	f-fg	fg-fg	SSGD
Train accuracy [%]	99.71	67.95	99.62	92.25	99.73	99.99
Test accuracy [%]	92.91	69.07	92.81	87.83	92.96	93.68

Table 2.2: Top training and testing accuracy obtained from the ResNet-18 experiment on CIFAR-10 dataset.

2.8 Conclusion

Dynamic sub-sampling updates mini-batches for every loss function evaluation, even along the same descent direction, allowing for a higher throughput of training data, when using smaller mini-batch sizes. The result is that the loss function is point-wise discontinuous or stochastic. As a result, it is challenging to conduct line searches when employing dynamic MBSS. We investigated the possibility of employing quadratic approximations to resolve learning rates. A carefully constructed and systematic empirical study revealed that it is essential to be selective about what information is enforced where along a descent direction. We investigated five approximations that differed only in the type of information, function value and directional derivative, and where they are enforced along a search direction, mainly at the start or end of a search interval.

The empirical results indicate that directional derivative information is more critical to enforce than function value information. In particular, enforcing the directional derivative information at the beginning of the search interval is more informative than the end-point. This mainly guarantees that the approximation optimizer is along the descent search direction, which also helps to reduce the variance of the solutions within this stochastic setting. It was found that enforcing any information for small batch sizes is useful. However, for deeper network tasks with only slightly larger mini-batch sizes, the distinction between enforcing fewer samples of selective information is much more beneficial than larger quantities of indiscriminate information. This implies that having less but the right type of information, in particular, directional derivatives, outperforms approximations using more information.

In this study, basic quadratic approximations proved to be competitive against a well-tuned scheduled SGD on CIFAR-10 using ResNet-18, and against scheduled SGD and Adam on MNIST using deep feedforward neural networks. Our experiments showed that quadratic approximation errors are manifested as biases and variances. We showed that reducing the variance error is more critical than reducing the bias error for training neural network problems. The variance could be reduced by using directional derivative information at the initial point of each approximation interval.

Chapter 3

GOALS: Gradient-Only Approximations for Line Searches Towards Robust and Consistent Training of Deep Neural Networks

3.1 Chapter overview

Mini-batch sub-sampling (MBSS) is favored in deep neural network training to reduce the computational cost. Still, it introduces an inherent sampling error, making the selection of appropriate learning rates challenging. The sampling errors can manifest either as a bias or variances in a line search. Dynamic MBSS re-samples a mini-batch at every function evaluation. Hence, dynamic MBSS results in point-wise discontinuous loss functions with smaller bias but larger variance than static sampled loss functions. However, dynamic MBSS has the advantage of having larger data throughput during training but requires the complexity regarding discontinuities to be resolved. This study extends the gradient-only surrogate (GOS), a line search method using quadratic approximation models built with only directional derivative information, for dynamic MBSS loss functions. We propose a gradient-only approximation line search (GOALS) with strong convergence characteristics with defined optimality criterion. We investigate GOALS's performance by applying it on various optimizers that include SGD, RMSprop and Adam on ResNet-18 and EfficientNet-B0. We also compare GOALS's against the other existing learning rate methods. We quantify both the best performing and most robust algorithms. For the latter, we introduce a relative robust criterion that allows us to quantify the difference between an algorithm and the best performing algorithm for a given problem. The results show that training a model with the recommended learning rate for a class of search directions helps to reduce the model errors in multimodal cases.

3.2 Introduction

In neural network training, choosing appropriate learning rates or learning rate schedules is non-trivial [Bengio, 2012, Goodfellow et al., 2016, Strubell et al., 2019]. As the neural network architectures become larger and more complex, the cost of training increases significantly [Brown et al., 2020]. The monetary, energy and CO_2 emission consequence of selecting a training strategy with significant performance variance, i.e. having near-optimal or poor training performance when combined with various optimizers, neural network architectures, and datasets, is noteworthy and important. Therefore, it becomes more and more important to formally quantify the robustness and consistency of a training approach instead of only quantifying its best performance. This also highlights the inherent Pareto optimal nature of selecting a training approach optimal for a specific application (specialist) or being adequate over a larger domain

of applications (generalist). In other words, the learning rate strategy selected needs to be robust enough that when it performs sub-optimally, the difference between its performance and the best performing approaches is limited. Incorporating this as a formal selection criterion improves the certainty with which an analyst can interpret the performance of an algorithm on a given problem without having to conduct additional exhaustive studies.

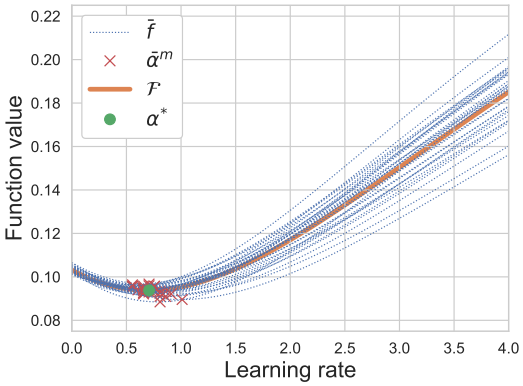
A natural consideration to resolve learning rates may be to consider line searches. Line searches are well-established in mathematical programming to efficiently resolve learning rates and identify descent directions. However, they require the underlying loss function to be convex or unimodal over an identified interval. This would be the case if full-batch training of machine learning and deep learning neural networks would be attenable. However, when conducting full-batch training, computational and memory requirements are untenable for practical training. This makes full-batch training ill-suited for DNNs on modern memory limited graphics processing unit (GPU) compute devices. As a result, the standard training procedure for machine learning and deep learning relies on mini-batch sub-sampling.

Mini-batch sub-sampling (MBSS) reduces the computational cost by using only a sub-sample of the training data at a time. This also provides a generalization effect [Masters and Luschi, 2018] by turning a smooth continuous optimization problem into a stochastic optimization problem [Robbins and Monro, 1951]. The stochastic or discontinuous nature of the loss function is due to the selected mini-batches' inherent sampling errors.

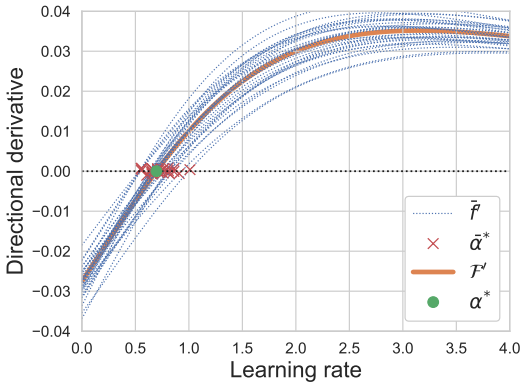
For line searches, the sampling errors manifest mainly in the form of bias or variance along a descent direction, depending on whether mini-batches are sub-sampled statically or dynamically [Chae and Wilke, 2019c, Kafka and Wilke, 2019a]. Static MBSS sub-samples a new mini-batch for every descent direction, while dynamic MBSS sub-samples a new mini-batch for every function evaluation. Hence, the loss function for static MBSS is continuous along a descent direction. The consequence is that the expected value of a static MBSS loss, has a small variance but a large bias compared to the expected or full-batch response. Conversely, the loss function for dynamic MBSS is point-wise discontinuous [Kafka and Wilke, 2019b, Chae and Wilke, 2019c]. The expected response of the dynamic MBSS loss has a small bias but a large variance compared to the expected or full-batch response [Kafka and Wilke, 2019b, Chae and Wilke, 2019c]. Consider Figure 3.1, which contrasts a full batched sampled loss function ($\mathcal{F}$ - orange) against a static ($\bar{f}$) and dynamic ($\tilde{f}$) sampled loss functions for fully-connected feedforward neural network [Mahsereci and Hennig, 2017] initialized with Xavier initialization [Glorot and Bengio, 2010]. Figure 3.1 (a) depicts 20 potential static MBSS loss functions. Each loss has zero variance but a large bias in this case. Figure 3.1 (c) depicts a dynamic MBSS loss function. It is clear that there is a large variance in the loss response, but the expected response has a lower bias since a mini-batch does not influence it in particular.

Line searches have been implemented for both static MBSS and dynamic MBSS loss functions. The main drawback of static MBSS is that it results in large biases in loss approximations and has been improved by applying sample variance reduction techniques [Friedlander and Schmidt, 2012, Bollapragada et al., 2018]. Meanwhile, attempts to resolve learning rates in the point-wise discontinuous loss approximations of dynamic MBSS include the probabilistic line search [Mahsereci and Hennig, 2017] and Gradient-Only Line Search that is Inexact (GOLS-I) [Kafka and Wilke, 2019b]. The probabilistic line search resolve learning rates by minimizing an approximation constructed using both function value and directional derivative information, while GOLS-I uses only directional derivative sign change information.

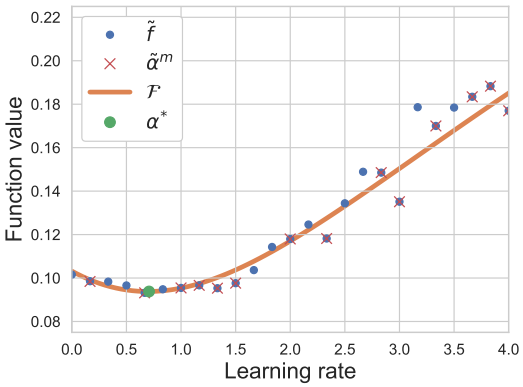
GOLS-I locates optima by searching for stochastic non-negative gradient projection points (SNN-GPP). These manifest as positive directional derivatives with a non-zero probability around a ball encapsulating SNN-GPPs. Any point around the SNN-GPP ball is taken and the directional derivative from the SNN-GPP to the point computed. An SNN-GPP along a descent direction merely manifests as a sign change from negative to positive. Importantly, a sign change from negative to positive is necessary and sufficient to identify an SNN-GPP, or alternatively stated, a minimizer as inferred solely from derivative information for this univariate case. This is empirically demonstrated in Figure 3.1.



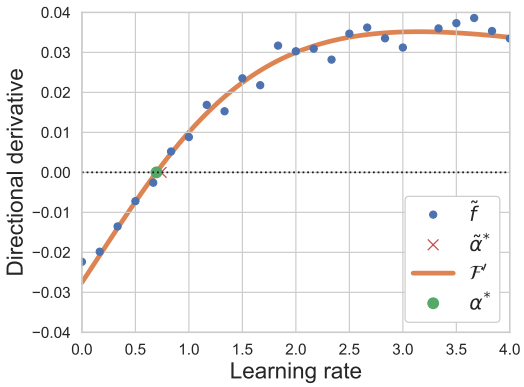
(a) Minimization with static MBSS



(b) Locating SNN-GPPs with static MBSS



(c) Minimization with dynamic MBSS



(d) Locating SNN-GPPs with dynamic MBSS

Figure 3.1: Illustration of finding local minima in (a) static and (c) dynamic MBSS loss functions, as well as locating SNN-GPPs using (b) static and (d) dynamic MBSS directional derivatives.

Learning rates resolved by locating minimizers or SNN-GPPs are equivalent for static MBSS loss functions as depicted by the minimizer solution, $\bar{\alpha}^m$, and SNN-GPP solution, $\bar{\alpha}^*$, in Figures 3.1 (a) and (b), respectively. However, for dynamic sampled loss functions, minimizers are identified over the entire domain, as shown in Figure 3.1 (c). In turn, SNN-GPPs are concentrated around the full batch solution, as shown in Figure 3.1 (d). SNN-GPPs indicate a lower bias to the expected full-batch minimizer than minimizers of SNN-GPPs resolved from the static MBSS loss.

A recent empirical study investigated how function value and directional derivative information help locate SNN-GPPs by comparing the resulting learning rates from various quadratic approximation models built from enforcing information in multiple ways [Chae and Wilke, 2019c]. Chae and Wilke [2019c] demonstrated that using only function value information resulted in learning rates with the larger variance due to the larger variance when predicting function values when conducting dynamic MBSS. Using only derivative information resulted in learning rates with smaller variance as the derivative information predicts more consistently when considering dynamic MBSS. Hence, gradient-only quadratic approximations result in stable and consistent learning rate predictions. Chae and Wilke [2019c] constructed derivative only approximations using only two directional derivative evaluations referred to as gradient-only surrogates (GOS). One evaluated at the origin, and the other at an "initial guess" learning rate along the descent direction. The directional derivative at the origin is strictly less than zero for descent directions. If the directional derivative at the initial guess is also less than zero, the initial guess learning rate is immediately accepted. In turn, if the directional derivative at the initial guess is positive, linear interpolation between the two points is performed to approximate the location of a directional derivative sign change. The proposed approach served as an initial investigation and proof of a "vanilla" line-search concept for only stochastic gradient descent (SGD). The "vanilla" directional-derivative-only approximation line search, proposed and investigated by Chae and Wilke [2019c], has no strong convergence characteristics, lacks a robust bracketing strategy and has not been demonstrated for descent directions strategies other than SGD. The contributions of this study include:

1. It extends the shortcomings of the vanilla directional-derivative-only approximation line search by proposing a line search with strong convergence characteristics. This is achieved by introducing a robust bracketing strategy to improve linear interpolation accuracy, referred to as the gradient-only approximation line search (GOALS). The bracketing strategy is based on a modified strong Wolfe condition [Wolfe, 1969, 1971] to isolate SNN-GPP. We essentially propose a conservative algorithm with strong convergence characteristics, which may result sacrifice the performance for convergence.

2. GOALS is demonstrated as a suitable line search strategy for descent direction approaches other than SGD, including RMSprop and Adam on deep neural network architectures with CIFAR-10 [Krizhevsky, 2009].

3. GOALS is compared to fixed learning rates, cosine annealing, GOLS-I, and GOS line search strategies on a shallow neural network architecture with MNIST [LeCun et al., 1998]. GOALS exhibits competitive results compared to other line search methods.

To compare the performance of different strategies, we introduced a relative robustness measure to quantify the differences between an algorithm and the best-performing algorithm for a given problem. In contrast to the traditional performance measure that only considers the best performance, the relative robustness measure considers all accounts, including poor performance. Hence, the criterion favors the strategy that performs well overall across different problems and optimizers rather than a problem-specific strategy.

The experimental results showed that our proposed algorithm GOALS ranked third and second for overall training and test relative robustness among ten strategies, led by GOS. Although GOS is not robust in convergence, it is more aggressive in training due to no curvature condition restricting its learning rates.

3.3 Background

In general, line searches can be employed to train deep neural networks to identify minimizers, first-order optimality candidate solutions (directional derivatives equal to 0) and SNN-GPPs. For convex functions, all three are equivalent. Several line searches which implement static MBSS have been presented, which take advantage of continuous loss functions that often assume convexity [Friedlander and Schmidt, 2012, Byrd et al., 2011, 2012a, Bollapragada et al., 2018, Kungurtsev and Pevny, 2018, Bergou et al., 2018, Mutschler and Zell, 2019, Yedida et al., 2021].

However, as illustrated in Figure 3.1, for dynamic MBSS loss functions, SNN-GPPs identify sensible solutions when compared to the full batch solution. Minimizers are hampered by local minima resulting in large variance, while first-order optimality candidate solutions may not exist for point-wise discontinuous loss functions. The present section summarizes several state-of-the-art sub-sampling and line search schemes applied to dynamic MBSS loss functions in machine learning literature. Firstly, we formalize dynamic MBSS and SNN-GPPs in Sections 3.3.1 and 3.3.2, respectively.

3.3.1 Dynamic mini-batch sub-sampling

Given weights $\boldsymbol{x}$, the function value computed with dynamic MBSS is expressed as

$$\tilde{L}(\boldsymbol{x}) = \frac{1}{|\mathcal{B}_{n,i}|} \sum_{b \in \mathcal{B}_{n,i}} \ell(\boldsymbol{x}; \boldsymbol{t}_b), \tag{3.1}$$

where $\ell(\boldsymbol{x}; \boldsymbol{t}_b)$ is computed using training samples in the sampled mini-batch, $\boldsymbol{t}_b$, with approximate gradient given by

$$\tilde{\boldsymbol{g}}(\boldsymbol{x}) = \frac{1}{|\mathcal{B}_{n,i}|} \sum_{b \in \mathcal{B}_{n,i}} \boldsymbol{\nabla} \boldsymbol{\ell}(\boldsymbol{x}; \boldsymbol{t}_b), \tag{3.2}$$

where i denotes the i-th function evaluation of the n-th iteration of a given algorithm. The loss function as a function of learning rate, α, along a given descent direction, $\boldsymbol{d}_n$, starting from $\boldsymbol{x}_n$ is given by:

$$\tilde{f}_n(\alpha) = \tilde{L}(\boldsymbol{x}(\alpha)) = \tilde{L}(\boldsymbol{x}_n + \alpha \boldsymbol{d}_n), \tag{3.3}$$

with the directional derivative, $\tilde{f}'_n$, given by

$$\tilde{f}'_n(\alpha) = \boldsymbol{d}_n^\intercal \tilde{\boldsymbol{g}}(\boldsymbol{x}_n + \alpha \boldsymbol{d}_n). \tag{3.4}$$

Dynamic MBSS loss functions are point-wise discontinuous functions with point-wise discontinuous gradient fields.

3.3.2 Gradient-only optimality criterion

Multiple local minima would be found when locating minimisers for discontinuous functions such as a dynamic MBSS loss function. Instead, we may opt to locate Non-Negative Gradient Projection Points (NN-GPPs) for which its gradient-only optimality criterion was specifically designed for deterministic discontinuous function [Wilke et al., 2013], given by

$$\boldsymbol{d}_n^\intercal \boldsymbol{\nabla} \boldsymbol{\mathcal{L}}(\boldsymbol{x}_{nngpp} + \alpha_n \boldsymbol{d}_n) \geq 0, \quad \forall \|\boldsymbol{d}_n \in \mathbb{R}^p\|_2 = 1, \quad \forall \alpha \in (0, \alpha_{max}], \tag{3.5}$$

for the 1-D case, along a given search direction, $\boldsymbol{d}_n$. NN-GPP is representative of a local optimum because no descent directions are allowed away from it. This is only possible at a critical point or a local minimum in a smooth and continuous function.

The NN-GPP definition is limited to deterministic discontinuous functions. Therefore, to accommodate stochastic discontinuous functions, the NN-GPP definition was generalized and extended to the Stochastic NN-GPP (SNN-GPP), given by

$$\boldsymbol{d}_n^\intercal \tilde{\boldsymbol{g}}(\boldsymbol{x}_{snngpp} + \alpha_n \boldsymbol{d}_n) \geq 0, \;\; \forall \|\boldsymbol{d}_n \in \mathbb{R}^p\|_2 = 1, \;\; \forall \alpha \in (0, \alpha_{max}], \;\; p(\boldsymbol{x}_{snngpp}) > 0, \tag{3.6}$$

with probability, $p(\boldsymbol{x}_{snngpp})$, greater than 0 [Kafka and Wilke, 2019b].

The difference between NN-GPP and SNN-GPP is that NN-GPP is a point where the signs of directional derivatives change in the deterministic setting. However, in the stochastic setting, a directional derivative sign change location may vary, depending on the instance of the sampled stochastic loss. Transferred to dynamic MBSS losses, this means that for each distinct mini-batch, $\mathcal{B}$, selected, we have a distinct location of a sign change. However, these remain bounded in a ball, B_ϵ, [Kafka and Wilke, 2019b] of a given loss landscape neighborhood. The size of B_ϵ is, among other factors, dependent on the variance in the stochastic loss function, which in dynamic MBSS losses is dependent on the mini-batch size. Hence, the larger the difference between individual samples, $\mathcal{B}_{n,i}$, the larger the size of the ball, B_ϵ. Notably, the SNN-GPP definition also generalizes to the NN-GPP, critical point and local minimum, as these are all SNN-GPPs with probability 1.

3.3.3 Line searches for dynamic MBSS loss functions

To the best of the author's knowledge, only three line search techniques have been proposed to resolve learning rates for dynamic MBSS loss functions, namely:

1. A probabilistic line search using Bayesian optimization with Gaussian surrogate models, built using both function value and directional derivative information [Mahsereci and Hennig, 2017].

2. The Gradient-only line search, Inexact (GOLS-I), locates SNN-GPPs along the search directions, using only directional derivative information [Kafka and Wilke, 2019b].

3. Proof of concept quadratic approximations [Chae and Wilke, 2019c].

All three line search methods showed competitive training performance for dynamic MBSS losses and outperformed various constant learning rates.

Notably, Chae and Wilke [2019c] the quality of function values and directional derivatives in the context of approximation-based line searches is investigated empirically. The quality of information used to produce approximations in dynamic MBSS losses was studied by constructing five types of quadratic approximation models using different information sampled at two locations (e.g. only function value, only directional derivative, both function value and directional derivative models) [Chae and Wilke, 2019c]. The results showed that using directional derivative information at the origin (starting point) of a line search is critical for constructing quality approximations, decreasing the variances in optimal learning rates. The two best performing models were

1. Derivative-only quadratic model: A quadratic approximation is built using two directional derivatives, values measured at the origin and another point along descent direction.

2. Mixed quadratic model: A quadratic approximation is built using the directional derivative measured at the origin and function values at both origin and another point.

Note that both quadratic approximation models proposed by Chae and Wilke [2019c] demonstrate "vanilla" algorithms without guaranteed convergence. Although the mixed-model has been investigated by Mahsereci and Hennig [2017] and Mutschler and Zell [2019] before, the derivative-only model has not been extended to a fully automated line search technique, which is the aim of this paper. Next, we discuss the heuristics and the corresponding shortcomings of the vanilla derivative-only approximation using the derivative-only model proposed in Chae and Wilke [2019c], extending in this study.

3.3.4 Gradient only surrogate (GOS)

Given a multivariate dynamic MBSS loss function, $\tilde{L}(\boldsymbol{x}_n)$, along a given descent direction, $\boldsymbol{d}_n$, $\tilde{f}(\alpha)$, we want to resolve the learning rate, α. The quadratic approximation model, $\hat{f}(\alpha)$, of

$\tilde{f}(\alpha)$ is given by

$$\hat{f}(\alpha) := k_1\alpha^2 + k_2\alpha + k_3 \approx \tilde{f}(\alpha), \tag{3.7}$$

where k_1, k_2 and k_3 are the constants to be computed. Similarly, the first-order derivative of the quadratic approximation, $\hat{f}'(\alpha)$, is a linear approximation given by

$$\hat{f}'(\alpha) := 2k_1\alpha + k_2 \approx \tilde{f}'(\alpha). \tag{3.8}$$

Note that $\hat{f}'(\alpha)$ is the derivative-only approximation proposed by Chae and Wilke [2019c], and is implemented throughout this paper. The approximation uses only directional derivative information. The constants at n-th iteration, $k_{1,n}$ and $k_{2,n}$ can be solved using a linear system of equations, constructed from two instances of Equation (3.8), given by

$$\begin{bmatrix} 2\alpha_{0,n} & 1 \\ 2\alpha_{1,n} & 1 \end{bmatrix} \begin{bmatrix} k_{1,n} \\ k_{2,n} \end{bmatrix} = \begin{bmatrix} \tilde{f}'_{0,n} \\ \tilde{f}'_{1,n} \end{bmatrix}, \tag{3.9}$$

where $\tilde{f}'_{0,n}$ and $\tilde{f}'_{1,n}$ denote the dynamic MBSS directional derivatives measured at $\alpha_{0,n}$ and $\alpha_{1,n}$ respectively, where $\alpha_{0,n} = 0$ is the current starting point and $\alpha_{1,n}$ is the initial guess ($\alpha_{1,n} > 0$). The approximate minimum, $\tilde{\alpha}^*_n$, is computed as $\alpha^*_n = -k_{2,n}/(2k_{1,n})$, where $\hat{f}'(\tilde{\alpha}^*_n) = 0$. For the implementation, we compute $\tilde{\alpha}^*_n$ in the closed-form as follows:

$$\alpha^*_n = \alpha_{0,n} - \tilde{f}'_{0,n}\frac{\Delta\alpha_n}{\Delta\tilde{f}'_n} = \alpha_{0,n} - \tilde{f}'_{0,n}\frac{\alpha_{1,n} - \alpha_{0,n}}{\tilde{f}'_{1,n} - \tilde{f}'_{0,n}}. \tag{3.10}$$

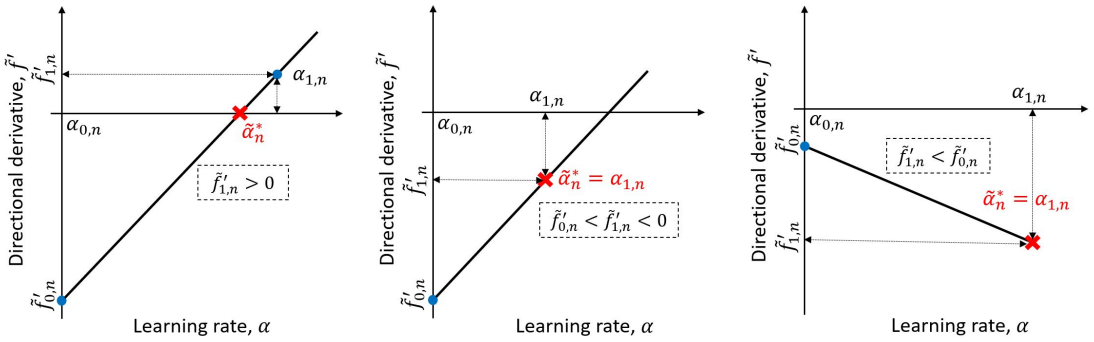
The heuristics of the vanilla line search algorithm using Equation (3.10), proposed by Chae and Wilke [2019c], are recalled here:

1. If $\tilde{f}'_{1,n} > 0$, as shown in Figure 3.2(a), we may perform bounded linear interpolations, Equation (3.10), to resolve learning rate, $\tilde{\alpha}^*_n$.
2. If $\tilde{f}'_{0,n} < \tilde{f}'_{1,n} < 0$, as shown in Figure 3.2(b), we can perform bounded extrapolation using Equation (3.10), but the prediction error is expected to be larger than the case of bounded interpolation. Hence, we immediately choose the initial guess as the resulted learning rate, i.e. $\tilde{\alpha}^*_n = \alpha_{1,n}$.
3. If $\tilde{f}'_{1,n} < \tilde{f}'_{0,n}$, as shown in Figure 3.2(c), it would be unwise to perform unbounded extrapolation for the same reason as in the case of bounded extrapolation. Therefore, we immediately accept the initial guess α^*, as the learning rate of the current iteration $\tilde{\alpha}^*_n = \alpha_{1,n}$.

The examples, studied by Chae and Wilke [2019c], used stochastic gradient descent (SGD) with the vanilla algorithm, $\boldsymbol{d}_n = -\tilde{\boldsymbol{g}}_n$. The initial guess, $\alpha_{1,n}$, for every iteration was chosen to be the inverse of L-2 norm of the search direction vector, $\alpha_{1,n} = \|\boldsymbol{d}\|_2^{-1}$. This means that $\alpha_{1,n}$ is adapted to the magnitude of the search direction vector, $\boldsymbol{d}_n$, to prevent overly aggressive (potentially unstable) training behavior, when $\|\nabla\tilde{\boldsymbol{\mathcal{L}}}_n\|_2 \approx 0$.

3.4 Robustness measure, R

Traditional performance measures only consider the top performances of strategies in DNNs. However, we observe rapid growth in the size of DNN problems, and the complexity of the problems also increases. This means that when we face an unseen problem, we might want to choose a more robust strategy across different problems and optimizers than a problem-specific top-performing strategy. Hence, we propose a new relative robustness measure that considers both poor and excellent performances of learning rate strategies across different problems and optimizers instead of considering only the top performances. We define the relative robustness measure for learning rate strategies as follows:



(a) Bounded Interpolation (b) Bounded extrapolation (c) Unbounded extrapolation

Figure 3.2: Illustration of three possible cases when implementing the vanilla line search algorithm using the derivative-only approximation: (a) bounded interpolation, when $\tilde{f}'_{1,n} > 0$, (b) bounded extrapolation when $\tilde{f}'_{0,n} < \tilde{f}'_{1,n} < 0$ and (c) unbounded extrapolation, when $\tilde{f}'_{1,n} < \tilde{f}'_{0,n}$.

Definition 3.4.1 (Relative robustness)**.** *The relative robustness, R, of a strategy, y, is computed by summing the absolute differences, $\psi_{y,h,o}$, in the strategy's accuracy, $\eta_{y,h,o}$, and the best accuracy of all strategies, $\eta^*_{h,o}$, for all optimizers, $O \ni o$, and all problems, $H \ni h$. Hence, the less the measure, R_y, is, the more robust the strategy is. The equation for R_y is given by*

$$R_y = \sum_{h \in H} \sum_{o \in O} \psi_{y,h,o}, \text{ where } \psi_{y,h,o} = |\eta^*_{h,o} - \eta_{y,h,o}|. \tag{3.11}$$

One may also compute a robustness measure, $R_{y,h}$, for a strategy, y and a specific problem, h while considering all optimizers, O. We will compare the training and test performance of our proposed algorithm, GOALS, against the other learning rate strategies based on the relative robustness measure throughout the paper.

3.5 Gradient-only approximation line search (GOALS)

This section proposes our line search strategy, gradient-only approximation line search (GOALS), using the quadratic derivative-only approximation, capable of locating the SNN-GPPs in the stochastic loss functions produced by dynamic MBSS. Unlike the vanilla algorithm, GOALS requires consecutive function evaluations to converge to an interval of sign changes for a given descent direction, $\boldsymbol{d}_n$, update.

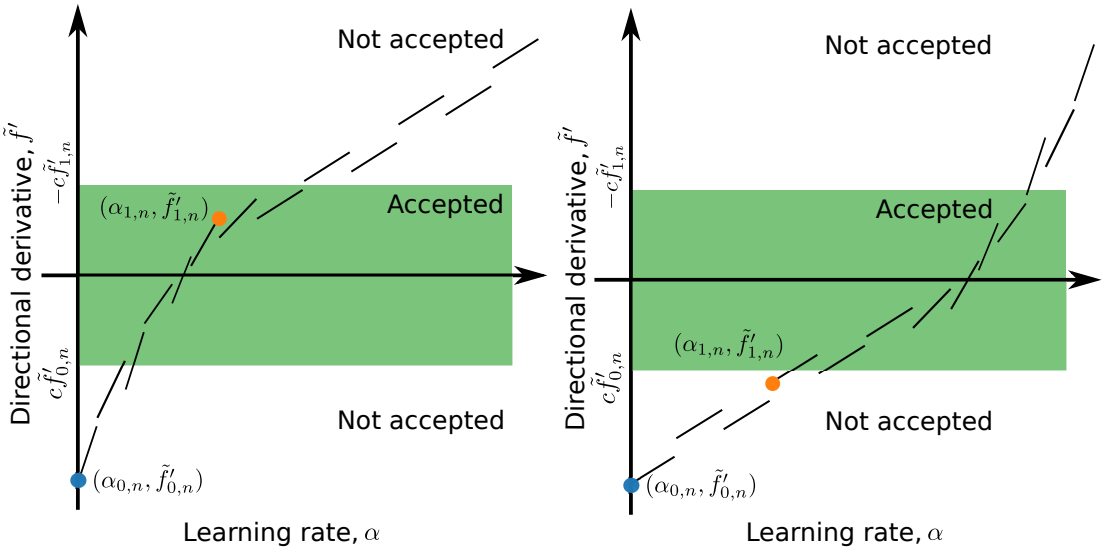
GOALS is comprised of the two main stages: 1) An immediate accept condition (IAC), and 2) a bracketing strategy. The IAC means that we accept the initial learning rate guess of the n-th iteration, $\alpha_{1,n}$, as the approximate solution, $\tilde{\alpha}^*_n$, when $\alpha_{1,n}$ falls within the accepted range set by the Wolfe condition [Wolfe, 1969, 1971]. If the IAC is not satisfied, the proposed bracketing strategy is used to locate SNN-GPPs.

3.5.1 Immediate accept condition (IAC)

The IAC aims to save computational cost. When the immediate accept condition (IAC) is satisfied, we immediately accept the initial guess, $\alpha_{1,n}$, and continue to the next search direction. The IAC is based on the Wolfe condition, consisting of two conditions: 1) Armijo condition and 2) Wolfe curvature condition. The Armijo condition ensures that the function value at the accepted learning rate decreases monotonically as outlined,

$$\tilde{f}_{1,n} \leq \tilde{f}_{0,n} + \omega \alpha_{1,n} \tilde{f}'_{0,n}. \tag{3.12}$$

Here, ω is a prescribed constant, often very small (e.g. $\omega = 10^{-4}$). Note that the Armijo condition limits the maximum learning rate by disallowing any increase in function value.



(a) Accepted (b) Not accepted

Figure 3.3: Illustration of immediate accept condition: (a) when the IAC (3.15) satisfies, the initial guess, $\alpha_{1,n}$, is accepted and (b) when the IAC (3.15) does not satisfy, the initial guess, $\alpha_{1,n}$, is not accepted.

The Wolfe curvature condition ensures that the directional derivative at the initial guess, $f'_{1,n}$, is less than the directional derivative at the starting point, $f'_{0,n}$. The condition is given by:

$$-\tilde{f}'_{1,n} \leq -c\tilde{f}'_{0,n}, \tag{3.13}$$

where $c \in (0, 1)$ a prescribed curvature constant. The Wolfe curvature condition limits the minimum learning rate based solely on directional derivative information. In contrast, the Armijo condition, Equation (3.12), requires function value information. Hence, it is not suitable for our derivative-only purpose. However, the Armijo condition is essential for limiting the maximum learning rates by not allowing growth in the function value. Therefore, we will implement the strong Wolfe condition as an alternative to Equations (3.12) and (3.13):

$$|\tilde{f}'_{1,n}| \leq c|\tilde{f}'_{0,n}|. \tag{3.14}$$

As a consequence of applying the strong Wolfe condition, we gain control over preventing overshooting, Equation (3.12), and undershooting, Equation (3.13), by changing the c constant.

Although we implement Equation (3.13) as the IAC throughout the paper, note that one could also independently control undershoot and overshoot limits by splitting c into two positive constants, c_1 and c_2, respectively, as follows:

$$c_1\tilde{f}'_{0,n} \leq \tilde{f}'_{1,n} \leq -c_2\tilde{f}'_{0,n}, \quad c_1, c_2 \in [0, 1). \tag{3.15}$$

If the IAC in Equation (3.14) is satisfied, there is just one directional derivative computation required to determine the learning rate along a descent direction, $\boldsymbol{d}_n$. Figure 3.3 illustrates cases of when the initial guess, $\tilde{f}'_{1,n}$, is either accepted (left) for satisfying the IAC condition, or not accepted (right) for not satisfying the condition.

However, if the initial guess is not accepted, we employ the bracketing strategy introduced in the next section to search for SNN-GPPs. Note that the larger the c value becomes, the larger the range of the IAC becomes. Algorithm 3 lists the pseudocode for the main GOALS algorithm with the IAC.

Algorithm 3: GOALS

Input: $\tilde{\boldsymbol{g}}_{0,n} = \tilde{\boldsymbol{g}}^*_{n-1}$, $\tilde{\alpha}^*_{n-1}$, $\boldsymbol{d}_n$, c, α_{min}, α_{max}, γ and ε
Output: $\tilde{\alpha}^*_n$, $\tilde{\boldsymbol{g}}^*_n$

1 Compute directional derivative at $\alpha_{0,n} \rightarrow \tilde{f}'_{0,n}$
`/* Magnitude check for avoiding numerical issues */`
2 **if** $|\tilde{f}'_{0,n}| < \varepsilon$ **then**
3 Recompute $\tilde{\boldsymbol{g}}^*_n$ for the next iteration.
4 **return** $\tilde{\alpha}^*_n := 0$, $\tilde{\boldsymbol{g}}^*_n$
`/* Setting the initial guess,` $\alpha_{1,n}$ `*/`
5 $\alpha_{1,n} \rightarrow \gamma$
6 Compute gradient and directional derivative at $\alpha_{1,n} \rightarrow \tilde{\boldsymbol{g}}_{1,n}, \tilde{f}'_{1,n}$
`/* Check whether the IAC satisfies */`
7 **if** $|\tilde{f}'_{1,n}| \leq c|\tilde{f}'_{0,n}|$ **then**
8 $\tilde{\alpha}^*_n := \alpha_{1,n}$ and $\tilde{\boldsymbol{g}}^*_n := \tilde{\boldsymbol{g}}_{1,n}$
9 **else**
10 $\tilde{\alpha}^*_n$,$\tilde{\boldsymbol{g}}^*_n := Bracketing(\alpha_{0,n}, \alpha_{1,n}, \tilde{f}'_{0,n}, \tilde{f}'_{1,n})$
11 **return** $\tilde{\alpha}^*_n$,$\tilde{\boldsymbol{g}}^*_n$

For every iteration, n, GOALS requires the gradient vector at the starting point, $\tilde{\boldsymbol{g}}_{0,n}$. Therefore, for $n > 0$, the resulting gradient from the previous iteration, $\tilde{\boldsymbol{g}}^*_{n-1}$, can be used for $\tilde{\boldsymbol{g}}_{0,n}$ in the next iteration. In line 2, the tolerance value, ε, ensures that $\tilde{f}'_{0,n}$ is a numerically positive value. Otherwise, we recompute the gradient at the same point with the resulted learning rate, $\tilde{\alpha}^*_n = 0$, using a new mini-batch, $\mathcal{B}$, and continue to the next iteration.

In line 5, the initial guess, $\alpha_{1,n}$, is set to be the default learning rate, γ, often the recommended learning rate for the chosen optimizer. Next, we compute the gradient vector, $\tilde{\boldsymbol{g}}_{1,n}$, and directional derivative, $\tilde{f}'_{1,n}$, at the initial guess, $\alpha_{1,n}$. In line 7, if the IAC satisfies, we choose the current learning rate, $\tilde{\alpha}^*_n = \alpha_{1,n}$, and the resulting gradient, $\tilde{\boldsymbol{g}}^*_n = \tilde{\boldsymbol{g}}_{1,n}$. If the IAC is not satisfied, we implement the bracketing strategy to compute $\tilde{\alpha}^*_n$ and $\tilde{\boldsymbol{g}}^*_n$. The bracketing strategy aims to minimize the model error using linear interpolation, introduced in the following section.

3.5.2 Bracketing strategy

The bracketing strategy aims to isolate an SNN-GPP inside an interval, $\mathcal{I} \in [\alpha_L, \alpha_U]$ with lower bound, α_L and upper bound, α_U, by updating $\mathcal{I}$ repeatedly, until the strong Wolfe condition, Equation (3.15) of $\tilde{\alpha}^*_n$ is satisfied. Once the directional derivative signs at α_L and α_U are found to brackets an SNN-GPP, we reduce the interval by applying the Regula-Falsi method [Gupta, 2019]. This is essentially a consecutive linear interpolation method, until $\tilde{\alpha}^*_n$ satisfies Equation (3.15). We provide the pseudocode for the bracketing strategy in Algorithm 4.

Algorithm 4: Bracketing

Input: $\alpha_{0,n}$, $\alpha_{1,n}$, $\tilde{f}'_{0,n}$, $\tilde{f}'_{1,n}$, $\boldsymbol{d}_n$, α_{min}, α_{max}, c and ε
Output: α^*_n, $\tilde{\boldsymbol{g}}^*_n$

1 Initialize $\alpha_L := \alpha_{0,n}$, $\tilde{f}'_L := \tilde{f}'_{0,n}$, $\alpha_U := \alpha_{1,n}$ and $\tilde{f}'_U := \tilde{f}'_{1,n}$
`/* Shifting the interval to larger learning rates */`
2 **while** $(\tilde{f}'_U < c\tilde{f}'_{0,n})$ *and* $(2\alpha_U < \alpha_{max})$ **do**
3 Update $\alpha_L := \alpha_U$ and $\tilde{f}'_L := \tilde{f}'_U$
4 $\alpha_U := 2(\alpha_U)$ and recompute $\tilde{\boldsymbol{g}}_U, \tilde{f}'_U$
5 $\tilde{f}'_{temp} := \tilde{f}'_U$
6 $\alpha_{temp} := \alpha_U$
`/* Shrinking the interval using linear interpolations */`
7 **while** $(\tilde{f}'_{temp} > -c\tilde{f}_{0,n})$ *and* $(\tilde{f}'_U\tilde{f}'_L < 0)$ *and* $(|\tilde{f}'_U - \tilde{f}'_L| > \varepsilon)$ **do**
8 $\alpha_{temp} = \frac{\alpha_L\tilde{f}'_U - \alpha_U\tilde{f}'_L}{\tilde{f}'_U - \tilde{f}'_L}$
9 Recompute $\tilde{f}'_{temp}$ at α_{temp}
10 **if** $\tilde{f}'_{temp}\tilde{f}'_L < 0$ **then**
11 $\tilde{f}'_U := \tilde{f}'_{temp}$
12 $\alpha_U = \alpha_{temp}$
13 **else**
14 $\tilde{f}'_L := \tilde{f}'_{temp}$
15 $\alpha_L = \alpha_{temp}$
16 $\alpha^*_n = \alpha_{temp}$
17 Compute gradient $\tilde{\boldsymbol{g}}^*_n$ at α^*_n
18 **return** α^*_n, $\tilde{\boldsymbol{g}}^*_n$

In line 1, we begin with initialization of the lower, α_L, and upper, α_U, bounds, and their respective directional derivatives, $\tilde{f}'_L$ and $\tilde{f}'_U$. In line 2, the interpolation interval grows by doubling α_U which is directly followed by α_L, until it reaches α_{max} or $\tilde{f}'_U \geq c\tilde{f}'_{0,n}$ is satisfied. In line 7, the size of the interpolation interval is reduced by consecutively performing linear interpolation until $\tilde{f}'_U \leq -c\tilde{f}_{0,n}$ is satisfied. The rest of conditions in line 7 ensures that linear interpolation steps with the Regula-Falsi method in lines 8-15 do not cause any numerical instabilities. The second term, $\tilde{f}'_U\tilde{f}'_L < 0$, ensures that the sign of the two directional derivatives are opposite to each other, and the last term, $|\tilde{f}'_U\tilde{f}'_L| < \epsilon$, provides the denominator of line 8 to be non-zero.

The Wolfe curvature conditions are divided into two sections, as shown in lines 2 and 7, to prevent this algorithm from searching infinitely for points that do not satisfy the Wolfe curvature conditions. Due to the stochastic nature of dynamic MBSS loss functions, it is not guaranteed that continually reducing learning rates would find a negative directional derivative with less magnitude than the directional derivative at the origin, $\tilde{f}'_{n,0}$. This implies that we can not assure that the first condition in line 2 is still met after the second condition in line 7 is satisfied. The risk associated with undershooting is lower than that of overshooting because overshooting may cause divergence in training, while undershooting, in the worst case, causes slower training. The flowchart of GOALS is shown in Figure 3.4.

3.5.3 Proof of convergence

Let us assume that the full-batch loss function, $\mathcal{L}(\boldsymbol{x})$, is a smooth coercive function of a weight vector, $\boldsymbol{x} \in \mathbb{R}^p$, so that we can replace $\mathcal{L}(\boldsymbol{x})$ with a Lyapunov function, $\Gamma(\boldsymbol{x})$ [Lyapunov, 1992]. The Lyapunov's global stability theorem states that a Lyapunov function, $\Gamma(\boldsymbol{x})$, results in $\lim_{n\to\infty} \boldsymbol{x}_n = 0$, $\forall\ \boldsymbol{x}_n \in \mathbb{R}^p$ under the following conditions:

1. Positivity: $\Gamma(\mathbf{0}) = 0$ and $\Gamma(\boldsymbol{x}) > 0$, $\forall\ \boldsymbol{x} \neq \mathbf{0}$
2. Coercive: $\lim_{\boldsymbol{x}\to\infty} \Gamma(\boldsymbol{x}) = \infty$

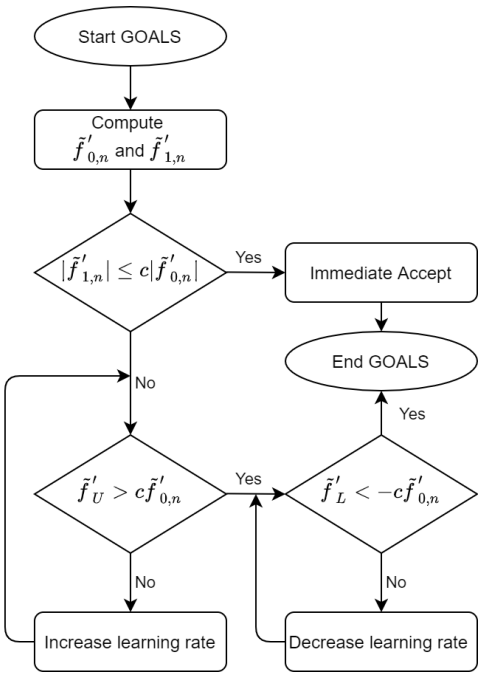


Figure 3.4: The flowchart of the GOALS line search strategy

3. Strict descent: $\Gamma(\mathcal{D}(\boldsymbol{x})) < \Gamma(\boldsymbol{x}),\ \forall\ \boldsymbol{x} \neq \mathbf{0}$

where $\mathcal{D}(\boldsymbol{x})$ is a weight update function given by

$$\boldsymbol{x}_{n+1} := \mathcal{D}(\boldsymbol{x}_n); \quad \mathcal{D} : \mathbb{R}^p \to \mathbb{R}^p. \tag{3.16}$$

It is proven by Wilke et al. [2013] that locating an NN-GPP along a strictly descending direction, $\boldsymbol{d}_n$, is equivalent to minimizing along $\boldsymbol{d}_n$ when $\mathcal{L}(\boldsymbol{x}_n + \alpha \boldsymbol{d}_n)$ is a smooth function. Therefore, locating NN-GPPs along descent directions in consecutive iterations of a training algorithm behaves like $\mathcal{D}(\boldsymbol{x})$. Similarly, for loss functions resulting from dynamic MBSS, $\tilde{\mathcal{L}}$, we assume as point-wise discontinuous coercive.

Hence, the Lyapunov's global stability theorem is relaxed for the expected Lyapunov function, $E[\Gamma(\boldsymbol{x})]$, where:

1. Expected positivity: $E[\Gamma(\mathbf{0})] = 0$ and $E[\Gamma(\boldsymbol{x})] > 0,\ \forall\ \boldsymbol{x} \neq \mathbf{0}$
2. Expected coercive: $\lim_{\boldsymbol{x}\to\infty} E[(\Gamma(\boldsymbol{x})] = \infty$
3. Expected strict descent: $E[\Gamma(\mathcal{D}(\boldsymbol{x}))] < E[\Gamma(\boldsymbol{x})],\ \forall\ \boldsymbol{x} \neq \mathbf{0}$

Subsequently, consecutively searching for SNN-GPPs along descent directions makes the training algorithm behave like $\mathcal{D}(\boldsymbol{x})$, which tends towards a ball, B_ϵ:

$$\lim_{\boldsymbol{n}\to\infty} \boldsymbol{x}_n = \{\boldsymbol{q} | \|\boldsymbol{q} - \boldsymbol{x}^*\| < \epsilon\} \in B_\epsilon \tag{3.17}$$

where B_ϵ is a ball function with the radius of ϵ, with the true optimum, $\boldsymbol{x}^*$, located at its center. Since our bracketing strategy searches for SNN-GPPs with weights, $\boldsymbol{x}_n \in \mathcal{B}_\epsilon$, respectively, along a strictly descending direction, $\boldsymbol{d}_n$,

$$|E[\Gamma(\boldsymbol{x}_{n+1})] - E[\Gamma(\boldsymbol{x}^*)]| < |E[\Gamma(\boldsymbol{x}_n)] - E[\Gamma(\boldsymbol{x}^*)]|, \tag{3.18}$$

and weights outside the ball, $\boldsymbol{x}_n \in \mathcal{B}'_\epsilon$, would eventually be inside the ball, $\boldsymbol{x}_n \in \mathcal{B}_\epsilon$, as $n \to \infty$.

3.6 Numerical study design

We conducted two sets of numerical studies to investigate the performance of the proposed learning rate algorithm, GOALS. First, we prepared GOALS with four different hyperparameter settings, namely, GOALS-1, 2, 3, 4. Their details are explained in Section 3.6.1. We compared them against the fixed learning rates, which are recommended learning rates for different optimizers, and the vanilla GOS on ResNet-18, and EfficientNet-B0 with the CIFAR-10 dataset for optimizers, including SGD, RMSProp, and Adam. We chose the batch size of 128 as implemented by Liu [2020]. From this experiment, we aim to investigate the followings:

1. Relative robustness of GOALS on the various optimizers that generate different descent directions;
2. Relative robustness of GOALS compared to the fixed learning rates and vanilla GOS;
3. Effect of various hyperparameter settings for GOALS;
4. Generalizability of the different learning rate strategies.

Note that as previously motivated in Section 3.4, we measured the performance of a strategy using the relative robustness measure in (3.11) since we were interested in a learning rate strategy that operates well over different problems and optimizers.

Generalization is an ability of a problem to perform well on unobserved inputs [Goodfellow et al., 2016]. Hence, to check generalizability, we measure the ratios of training to test accuracies. When the ratios approach one, it implies that the relative discrepancy between the training accuracy and test accuracy is small, which indicates the problem generalizes well.

Second, we conducted a hyperparameter study for GOALS using only SGD on a shallower DNN, N-II, implemented by Mahsereci and Hennig [2017]. We tested the four hyperparameter settings for GOALS and the most robust setting on our relative robustness measure was tested against other learning rate strategies. These included fixed learning rates, vanilla GOS [Chae and Wilke, 2019c], GOLS-I [Kafka and Wilke, 2019a], cosine annealing with warm restarts [Loshchilov and Hutter, 2016]. We tested multiple batch sizes of 10, 100, 200, and 1000. The small batch size of 10 allows us to investigate the behaviors of strategies when the information is critically sparse. We only used SGD as the optimizer for this experiment because SGD is sensitive to different learning rates as we compare strategies. We aimed to investigate the following in the second experiment:

1. Relative robustness of GOALS compared to the fixed learning rates and vanilla GOS on different batch sizes;
2. Effect of various hyperparameter settings for GOALS;
3. Generalizability of the different learning rate strategies.

3.6.1 Hyperparameter settings of GOALS

GOALS requires three hyperparameters to be selected:

1. The initial learning rate, $\alpha_{0,1}$;
2. The curvature hyperparameter, c;
3. A decision on whether we want to use the last learning rate for the following initial learning rate, $\alpha_{0,n} = \alpha^*_{0,n-1}$. The opposite of this would be to reset the next initial learning rate to the default initial learning rate, $\alpha_{0,n} = \alpha_{0,1}$.

Table 3.1 lists the four different settings of hyperparameters for GOALS. While all four settings keep the curvature hyperparameter, c, identical and large, the initial learning rates, $\alpha_{0,1}$, are either the recommended learning rates, γ, for the selected optimizer or the inverse of the L-2 norm of the search direction, $1/\|\boldsymbol{d}_n\|$. Note that the hyperparameter setting closest to the vanilla GOS is GOALS-4 allowing for direct comparison of performance between them.

Algorithms	Settings	$\alpha_{0,1}$	$\alpha_{0,n} = \alpha^*_{0,n-1}$	c
GOALS	GOALS-1	γ	No	0.9
	GOALS-2	γ	Yes	0.9
	GOALS-3	$1/\|\boldsymbol{d}_n\|$	Yes	0.9
	GOALS-4	$1/\|\boldsymbol{d}_n\|$	No	0.9
GOS	-	$1/\|\boldsymbol{d}_n\|$	No	-

Table 3.1: Comparison between the settings of vanilla GOS and GOALS that are tested in this paper. We choose the initial learning rates for the first iteration, $\alpha_{0,1}$, the curvature hyperparameter, c, and decide whether we want to use the final learning rate as the next initial learning rate, $\alpha_{0,n} = \alpha^*_{0,n-1}$.

3.6.2 Numerical study 1 setup

For the first numerical study, we investigate the performance of GOALS for different architectures and optimizers. It is compared against GOS and fixed learning rates for various optimizers' search directions: SGD, RMSProp, and Adam. We chose ResNet-18 [He et al., 2016] and EfficientNet-B0 [Tan and Le, 2019], which are implemented by Liu [2020] in PyTorch [Paszke et al., 2019], for the test DNN models and the CIFAR-10 dataset [Krizhevsky, 2009] for this experiment. The details of the dataset are shown in Table 3.2. The chosen mini-batch size, $|\mathcal{B}|$, for this numerical study is 128.

The following learning rate strategies were trained for 350 epochs, repeated five times: fixed learning rate, GOS, GOALS-1, GOALS-2, GOALS-3, and GOALS-4. The fixed learning rates, γ, for SGD, RMSProp, and Adam are 0.01, 0.01, and 0.001, respectively, which are the default values provided by PyTorch [Paszke et al., 2019] and TensorFlow [Abadi et al., 2015].

Note that because we adopt dynamic MBSS, a different mini-batch for every function evaluation, for the experiments, the fixed number of epoch also means the fixed number of gradient computations in training. Hence, some strategies may have fewer search directions when more gradient evaluations are required for each search direction or iteration.

Datasets	Classes	Input sizes	Training samples	Test samples
MNIST [LeCun et al., 1998]	10	28×28	6×10^4	1×10^4
CIFAR-10 [Krizhevsky, 2009]	10	32×32	5×10^4	1×10^4

Table 3.2: Descriptions of datasets used in the numerical study

3.6.3 Numerical study 2 setup

For the second numerical study, we first examine the performance of GOALS with different hyperparameter settings: GOALS-1, 2, 3, and 4 to choose the most robust hyperparameter setting, based on the relative robustness measure. Next, we test GOALS with the best hyperparameter setting against various learning rate strategies such as GOS, GOLS-I, fixed learning rate methods, cosine annealing with warm restarts [Loshchilov and Hutter, 2016] with different mini-batch sizes. We restrict ourselves to SGD directions on shallower neural network architecture.

We used similar neural network training problems to those proposed by Mahsereci and Hennig [2017], namely training on a fully-connected feedforward neural network problem, N-II. This network's architecture involves fully connected layers with three hidden layers, $n_{input} - 1000 - 500 - 250 - n_{output}$. Hence, this architecture has shallower networks compared to the test problems in Numerical study 1. It contains the tanh activation functions, mean square loss, and Xavier initialization [Glorot and Bengio, 2010]. The dataset used for the problem is MNIST in Table 3.2.

The descriptions of the learning rate strategies compared against GOALS are listed as follows:

1. Fixed learning rates: we test five sets of fixed learning rates, $\alpha = 10^{-3}$, 10^{-2}, 10^{-1}, 10^{0}

and, 10^1;

2. The cosine annealing scheduler with warm restarts [Loshchilov and Hutter, 2016]: starting learning rates used are $\alpha = 10^{-1}$ and 10^0, the initial restart period, the multiplying factor is chosen to be $T_0 = 1$ epoch, and $T_{mult} = 2$;

3. The gradient-only line search that is Inexact (GOLS-I) [Kafka and Wilke, 2019b];

4. The vanilla gradient-only surrogate/approximation (GOS) line search [Chae and Wilke, 2019c].

5. The gradient-only approximation line search (GOALS) allows various hyperparameter settings. For the comparison, we choose GOALS-4 in Table 3.1.

This makes ten strategies in total. The training is limited in the number of directional derivative computations per training run, and the limit is 4×10^4. The mini-batch sizes chosen were $|\mathcal{B}| \in \{10, 100, 200, 1000\}$. We include the batch size of 10 to investigate how the strategies behave when insufficient information is provided for them. As mentioned earlier, we select the SGD direction as the search directions, computed using the same mini-batch size, $|\mathcal{B}|$. For each setting, we take ten runs for generating results.

3.7 Results of numerical study

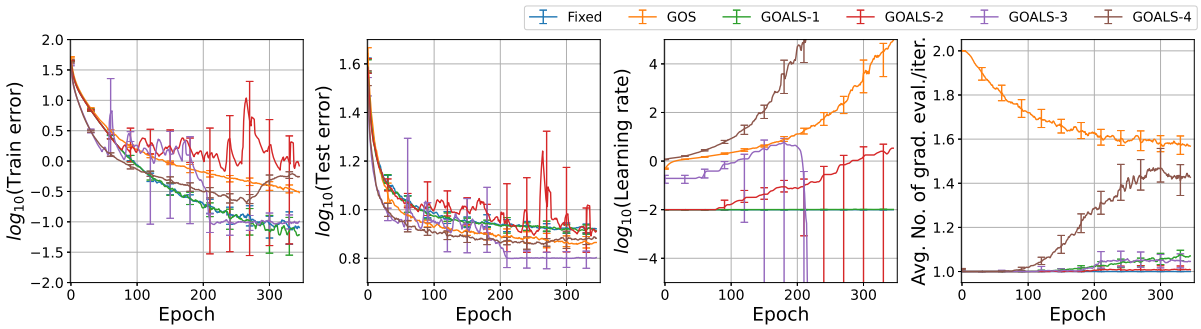
3.7.1 Results of numerical study 1

Figures 3.5 and 3.6 show the results for SGD, RMSprop and Adam, respectively. For each optimizer, the 5-step simple moving average values of the training errors, test errors, learning rates, and the number of gradient computations are plotted along 350 epochs using error bars on a $\log_{10}$ scale. The lower errors and upper errors represent the minimum and the maximum errors of the five runs. Note that dynamic MBSS requires a new mini-batch for every function evaluation. Hence, the larger the number of function evaluations computed per iteration, the fewer search direction updates per epoch.

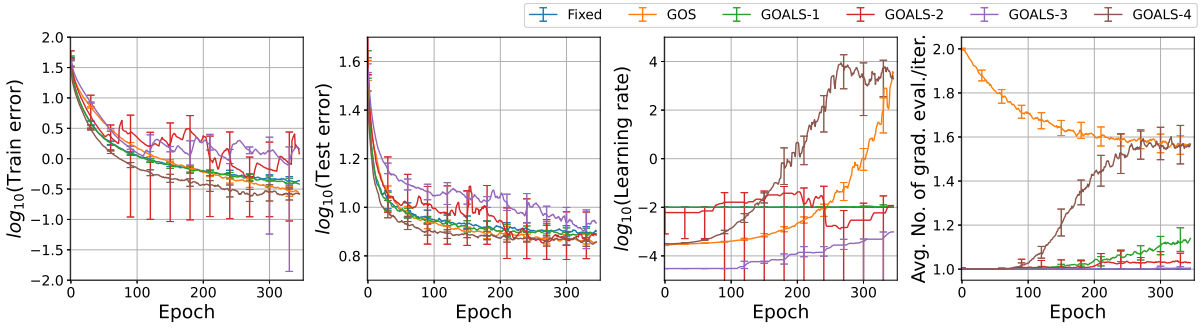
A common phenomenon observed in most of the results in Figures 3.5 and 3.6 is that the average learning rates of both GOS and GOALS increase as the epoch increases. This happens because the directional derivative at the origin, $\tilde{f}'_{0,n}$, decreases throughout training. This means that the average number of gradient computations for GOALS may increase over epochs to satisfy the curvature condition on the flatter domains of the functions. On the other hand, the average number of gradient computations for GOS decreases since the chance of observing a directional derivative at $\alpha_{1,n}$, $\tilde{f}'_{1,n}$ less than the initial directional derivative, $\tilde{f}'_{0,n}$, grows. In this case, it is the immediate accept condition (IAC). Hence, we accept $\alpha_{1,n}$ as the final learning rate, α^*_n.

The ResNet results shown in Figure 3.5 indicate that the initial convergence rate in GOS's training is slightly lower than for the fixed learning rates and GOALS. This is because GOS has a small learning rate initially since the inverse of the norm of search direction is small and does not extend the learning rate to be inside the ball like GOALS. Figure 3.5(a) shows that although the training error of GOS is relatively high, its test error is one of the lowest. The performance of GOALS-1 is similar to that of the fixed learning rate since the initial learning rate, $\alpha_{0,n} = \gamma$, happens to satisfy the Wolfe condition most of the time. Hence, note that there is a slightly increasing number of gradient evaluations only at the end of the training. GOALS-2 shows a large variance in the performance since using previous learning rates may cause a large model error as the previous learning rate might be far from the ball. Hence, it is more challenging to find the SNN-GPPs for possibly multimodal distributions of sign changes.

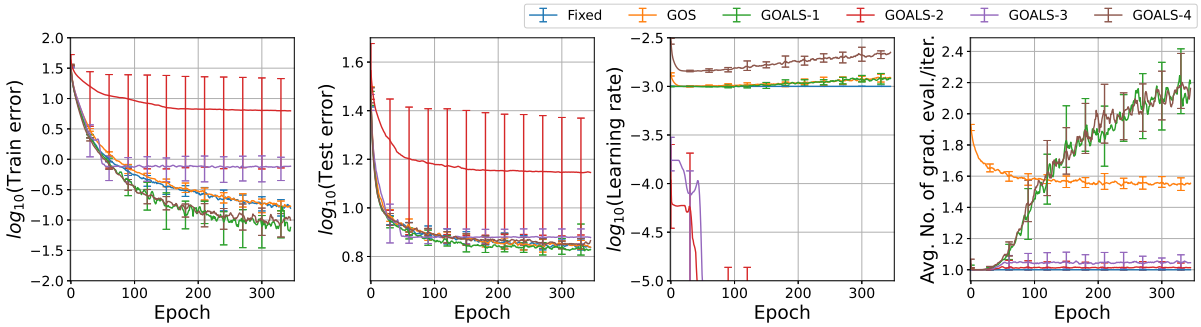
Note that the lower limit of GOALS' learning rates is softly bounded since we satisfy the lower curvature condition before satisfying the upper independently. Consequently, when the model error is large due to using the previous learning rates as initial guesses, it may allow learning rates to be numerically zero. For the same reason, we also observe the phenomenon



(a) SGD



(b) RMSprop



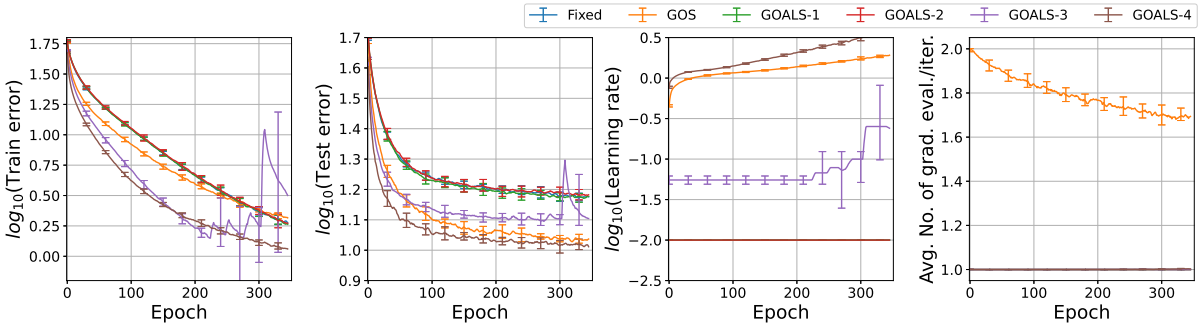
(c) Adam

Figure 3.5: Comparisons of the performances of (a) SGD, (b) RMSprop, (c) Adam between with and without GOALS applied, tested on ResNet-18 for the CIFAR-10 dataset, the results are averaged over five runs and smoothened out with moving average over five epochs. From left to right, it presents the training errors, test errors, learning rates on the $\log_{10}$ scale, and the average number of gradient evaluations per every iteration.

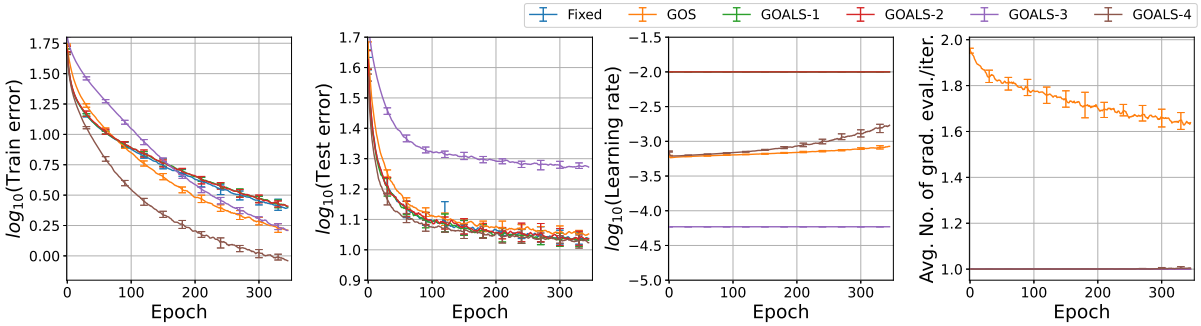
of diminishing learning rate in GOALS-3. However, it starts with larger learning rates and a steeper convergence rate in training, and it also shows the lowest test error for SGD. Although GOALS-4 has a similar hyperparameter setting as GOS, it shows quicker convergence in both training and test. The growth rate of its learning rate is also faster than GOS, and the average number of gradient evaluations increases faster than the other hyperparameter settings.

Figure 3.5(b) shows that the ResNet-18 results for RMSprop show that GOALS-4 has the lowest training and test errors, while both GOALS-2 and GOALS-3 show large fluctuations in learning rates. GOALS-1 and the fixed learning rate perform similarly since the recommended learning rate approximates SNN-GPPs well.

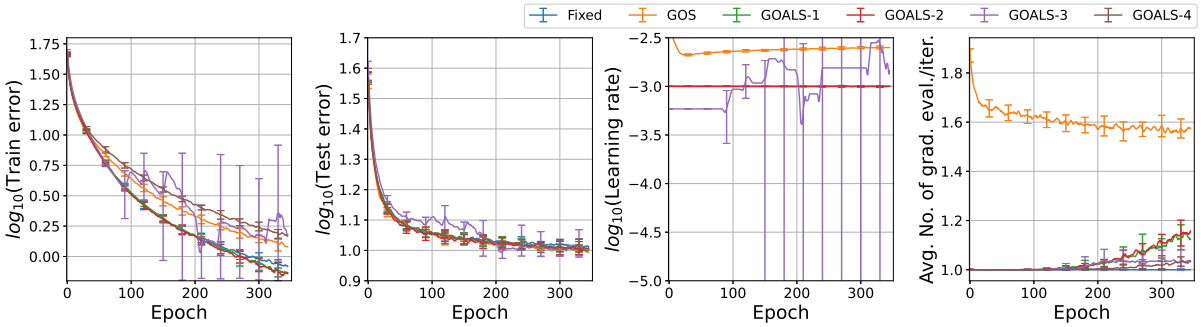
The ResNet-18 results for Adam in Figure 3.5(c) show that while both GOALS-2 and GOALS-3 again perform poorly, both GOS and the fixed learning rate perform similarly. Note that GOALS-1 and GOALS-4 show the best training and test performance. Also, note that the average numbers of gradient computations for GOALS-1, 4 increase as the epoch increases, and it shows the largest values among the optimizers. This means the recommended learning rate,



(a) SGD, $\alpha = 0.01$



(b) RMSProp



(c) Adam

Figure 3.6: Comparisons of the performances of (a) SGD, (b) RMSProp, (c) Adam between with and without GOALS applied, tested on EfficientNet-B0 for the CIFAR-10 dataset, the results are averaged over five runs and smoothened out with moving average over five epochs. From left to right, it presents the training errors, test errors, learning rates on the $\log_{10}$ scale, and the average number of gradient evaluations per every iteration.

γ, for Adam requires more adjustments to satisfy the curvature condition close to the end of training.

The EfficientNet-B0 results shown in Figures 3.6(a) and (b) indicate that SGD and RMSProp are mostly unaffected by GOALS-1 and GOALS-2. This is because their default learning rates mostly satisfy the curvature condition. Hence, the IAC continues through the whole training. However, Adam with GOALS affected the learning rates close to the end of training with slightly lower training and test errors. GOALS-3 for SGD, shown in Figure 3.6(a), has a high initial convergence rate. However, the large variance in learning rates resulted in fluctuations in the training error. GOALS-4 shows the lowest training and test errors for both SGD and RMSProp followed by GOS. Yet, it requires fewer gradient computations. Figure 3.6(c) shows the average numbers of gradient computation for both GOALS-1 and GOALS-2 increase for Adam, unlike SGD and RMSProp. This results in more adjustments in learning rates and improved train accuracies compared to the fixed learning rate.

Tables 3.3 and 3.4 present the top average training and test accuracies, and the performance

Models	Optimizers	Strategies	Train acc. [%]	Diff., $\psi_{y,h,o}$	Test acc. [%]	Diff., $\psi_{y,h,o}$	Te./Tr.
ResNet-18	SGD	Fixed	99.94	0.02	91.88	1.9	0.919
		GOS	99.72 (-0.22)	0.24	92.93 (+1.05)	0.85	0.932
		GOALS-1	**99.96 (+0.02)**	0	91.96 (+0.08)	1.82	0.92
		GOALS-2	99.43 (-0.51)	0.53	92.33 (+0.45)	1.45	0.929
		GOALS-3	99.93 (-0.01)	0.03	**93.78 (+1.90)**	0	0.938
		GOALS-4	99.8 (-0.14)	0.16	93.02 (+1.14)	0.76	0.932
	RMSProp	Fixed	99.64	0.15	92.37	0.65	0.927
		GOS	99.78 (+0.14)	0.01	**93.02 (+0.65)**	0	0.932
		GOALS-1	99.65 (+0.01)	0.14	92.59 (+0.12)	0.43	0.929
		GOALS-2	99.56 (-0.08)	0.23	**93.02 (+0.65)**	0	0.934
		GOALS-3	99.25 (-0.39)	0.54	92.24 (-0.13)	0.78	0.929
		GOALS-4	**99.79 (+0.15)**	0	93.01 (+0.64)	0.01	0.932
	Adam	Fixed	99.87	0.09	93.3	0.28	0.934
		GOS	99.86 (-0.01)	0.1	93.23 (-0.07)	0.35	0.934
		GOALS-1	**99.96 (+0.09)**	0	**93.58 (+0.28)**	0	0.936
		GOALS-2	93.79 (-6.08)	6.17	86.08 (-7.22)	7.5	0.918
		GOALS-3	99.31 (-0.56)	0.65	92.53 (-0.77)	1.05	0.932
		GOALS-4	99.94 (+0.07)	0.02	93.16 (-0.14)	0.42	0.932
	$R_{y,h}$	Fixed	-	0.26	-	2.83	-
		GOS	-	0.35	-	1.2	-
		GOALS-1	-	**0.14**	-	2.25	-
		GOALS-2	-	6.93	-	8.95	-
		GOALS-3	-	1.22	-	1.83	-
		GOALS-4	-	0.18	-	**1.19**	-
	Avg. Te./Tr.	Fixed	-	-	-	-	0.927
		GOS	-	-	-	-	0.933
		GOALS-1	-	-	-	-	0.928
		GOALS-2	-	-	-	-	0.927
		GOALS-3	-	-	-	-	0.933
		GOALS-4	-	-	-	-	0.932

Table 3.3: Top average training and test accuracies over the five runs tabulated for optimizers, including SGD, RMSprop and Adam, with the fixed recommended learning rates, vanilla GOS, and GOALS with various settings on ResNet-18. The differences in performance compared to the fixed learning rate are given inside the brackets. It measures the relative robustness, $R_{y,h}$, by computing summing the differences, $\psi_{y,h,o}$, between the performance and the best one from the same optimizer. The ratios of test to training accuracies are given in the last column, and the average ratios for each optimizer are computed in the last row. The highest train, test accuracies, and the lowest robustness measures are indicated in bold.

differences, $\Psi_{y,h,o}$ for them, and the ratios of the training to test accuracies for Numerical study 1. The ratio measures the generalization of each learning rate strategy. The closer the ratio becomes to one, the smaller discrepancy between the training accuracy and test accuracy becomes. This implies that the strategy generalizes well for the optimizer and problem. The tables also provide the relative robustness measure, $R_{y,h}$, over different optimizers, o, by summing the differences, $\Psi_{y,h,o}$, and the average ratios of test accuracies to training accuracies over the optimizers.

Table 3.3 shows the results of ResNet-18. The $R_{y,h}$ values indicate that GOALS-1 and GOALS-4 are the most robust strategies for training and test, respectively, over the optimizers for ResNet-18 among the six strategies. Concerning the ratios, GOS and GOALS-3 have the highest generalizability as their ratios are the closest ones, followed by GOALS-4. On the other hand, the lowest ratio, shown by GOALS-2, indicates that GOALS-2 experiences more overfitting compared to others on ResNet-18.

Table 3.4 shows the results of EfficientNet-B0. The $R_{y,h}$ values indicate that GOALS-4 is the most robust strategy for both training and test, over the optimizers for EfficientNet-B0 among the six strategies. GOS shows the average ratio closest to one. Hence, GOS is again the most generalizing strategy, followed by GOALS-4. Note that it is more challenging to generalize on EfficientNet-B0 compared to ResNet-18. GOALS-3 shows the lowest ratio of test to training

Models	Optimizers	Strategies	Train acc. [%]	Diff., $\psi_{y,h,o}$	Test acc. [%]	Diff., $\psi_{y,h,o}$	Te./Tr.
EfficientNet-B0	SGD	Fixed	98.18	0.7	85.37	4.48	0.87
		GOS	97.94 (-0.24)	0.94	89.44 (+4.07)	0.41	0.913
		GOALS-1	98.25 (+0.07)	0.63	85.39 (+0.02)	4.46	0.869
		GOALS-2	98.15 (-0.03)	0.73	84.97 (-0.4)	4.88	0.866
		GOALS-3	98.64 (+0.46)	0.24	87.8 (+1.43)	2.05	0.89
		GOALS-4	**98.88 (+0.7)**	0	**89.85 (+4.48)**	0	0.909
	RMSProp	Fixed	97.6	1.53	89.44	0.15	0.916
		GOS	98.43 (+0.83)	0.7	89.0 (-0.44)	0.59	0.904
		GOALS-1	97.51 (-0.09)	1.62	89.55 (+0.11)	0.04	0.918
		GOALS-2	97.46 (-0.14)	1.67	89.42 (-0.02)	0.17	0.918
		GOALS-3	98.4 (+0.8)	0.73	81.51 (-7.93)	8.08	0.828
		GOALS-4	**99.13 (+1.53)**	0	**89.59 (+0.15)**	0	0.904
	Adam	Fixed	99.21	0.15	89.96	0.34	0.907
		GOS	98.85 (-0.36)	0.51	90.22 (+0.26)	0.08	0.913
		GOALS-1	99.34 (+0.13)	0.02	90.22 (+0.26)	0.08	0.908
		GOALS-2	**99.36 (+0.15)**	0	90.13 (+0.17)	0.17	0.907
		GOALS-3	99.0 (-0.21)	0.36	**90.3 (+0.34)**	0	0.912
		GOALS-4	98.57 (-0.64)	0.79	90.18 (+0.22)	0.12	0.915
	$R_{y,h}$	Fixed	-	2.38	-	4.97	-
		GOS	-	2.15	-	1.08	-
		GOALS-1	-	2.27	-	4.58	-
		GOALS-2	-	2.4	-	5.22	-
		GOALS-3	-	1.33	-	10.13	-
		GOALS-4	-	**0.79**	-	**0.12**	-
	Avg. Te./Tr.	Fixed	-	-	-	-	0.898
		GOS	-	-	-	-	0.91
		GOALS-1	-	-	-	-	0.898
		GOALS-2	-	-	-	-	0.897
		GOALS-3	-	-	-	-	0.877
		GOALS-4	-	-	-	-	0.909

Table 3.4: Top average training and test accuracies over the five runs tabulated for optimizers, including SGD, RMSProp and Adam, with the fixed recommended learning rates, vanilla GOS, and GOALS with various settings on EfficientNet-B0. The differences in performance compared to the fixed learning rate are given inside the brackets. It measures the relative robustness, $R_{y,h}$, by computing summing the differences, $\psi_{y,h,o}$, between the performance and the best one from the same optimizer. The ratios of test to training accuracies are given in the last column, and the average ratios for each optimizer are computed in the last row. The highest train, test accuracies, and the lowest robustness measures are indicated in bold.

Strategies	Training R_y	Test R_y	Avg. Te./Tr.
Fixed	2.64	7.8	0.913
GOS	2.5	2.28	0.922
GOALS-1	2.41	6.83	0.913
GOALS-2	9.33	14.17	0.912
GOALS-3	2.55	11.96	0.905
GOALS-4	**0.97**	**1.31**	0.921

Table 3.5: The training and test relative robustness, R_y, for different strategies are given by summing the training and test relative robustness, $R_{y,h}$, over different problems, given in Tables 3.3 and 3.4. The average ratios of training to test accuracies for each strategy is given as the average values of the ratios (Te./Tr.) from the two problems in Tables 3.3 and 3.4. The lowest train and test accuracies robustness measures are indicated in bold.

accuracies, meaning that GOALS-3 experiences more overfitting.

Table 3.5 computes the overall training and test robustness measures, R_y, and the average ratios of test to training accuracies over the two problems, ResNet-18 and EfficientNet-B0. Hence, the relative robustness, R_y, for a strategy is computed as the sum of $R_{y,h}$ from Tables 3.3, 3.4. Both training and test R_y is the lowest with GOALS-4, followed by GOS. However, GOS has the highest generalizability ratio, closely followed by GOALS-4. According to the results, GOALS-4 is the most robust hyperparameter setting for GOALS in DNNs, while GOALS-3 is the least robust in testing and generalization, leading to overfitting. The only difference between the GOALS-3 and GOALS-4 is that GOALS-4 resets the next initial learning rate to the default initial learning rate.

From the results of Numerical study 1, we discovered that using the previous resultant learning rates from the previous iteration, α^*_{n-1}, as the next initial guess, $\alpha_{0,n}$, is not a reliable plan for using quadratic approximations. When for a highly non-linear problem, we expect a large discrepancy in the shapes of the approximation in different iterations. Hence, it is much more conservative to start the initial guess with the recommended learning rate, γ, for the chosen optimizer at every iteration. This phenomenon was apparent when the optimizer was ADAM.

As a result, we recommend the user employ the GOALS-4 hyperparameter setting, as it resets the initial guess as the fixed recommended learning rates, γ, at every iteration. Among the different hyperparameter settings, GOALS-4 had the best relative robustness measures, R_y, for both training and test accuracies, and it also had the best generalizability among the different hyperparameter settings. This means that GOALS-4 is least likely to fail to an unseen problem.

GOALS-4 tended to show more aggressive learning rates followed by GOS, which led to better performance than GOALS. However, this is because GOS does not have any convergence proof enforced, and it is not robust in terms of convergence.

3.7.2 Results of numerical study 2

This section studies GOALS on a shallower DNN problem, N-II. We start with the hyperparameter study of four different settings, as shown in Table 3.1. We expect GOALS-4 to outperform other settings because the setting does not use the previously resolved learning rate, α^*_{n-1}, as the next initial guess, $\alpha_{0,n}$. This reduces the model error based on the previous study. Additionally, we use the $1/\|\boldsymbol{d}_n\|_2$ value as the initial guess, which elongates as the magnitudes of gradients reduce. This helps to increase the model accuracy.

Next, we determine the most robust GOALS setting based on the relative robustness measure and compare it to nine other strategies only on SGD, which is an optimizer that is sensitive to the choice of learning rates. The results we obtain from this numerical study help us to determine which strategies are robust in training DNNs.

Performance test for different hyperparameter settings

Figure 3.7 shows the training error, test error, and learning rates for the various hyperparameter settings on the N-II architecture with MNIST. Note that the training and test errors for MNIST are plotted on the $\log_{10}$ scale, averaged over ten runs.

GOALS-1, which starts every iteration with the initial learning rate of 0.01, did not modify the learning rates. This means that the fixed recommended learning rate of SGD continuously satisfied the curvature condition, resulting in the IAC condition. Hence, it would perform almost exactly like the fixed learning rate of 0.01. Both GOALS-2 and GOALS-3 reuse the previously resolved learning rates. As a result, both settings experience significant approximation errors, and the learning rates are unstable, often approaching zeros.

On the other hand, GOALS-4, whose setting is similar to GOALS-1 except for the initial guess equals $1/\|\boldsymbol{d}_n\|_2$, outperforms other hyperparameter settings, and the learning rates are close to the vanilla GOS. The differences in learning rates tend to increase as the larger batch sizes grow. GOALS-4 shows its best performance when its learning rates are closest to GOS. This is when the batch size is 100.

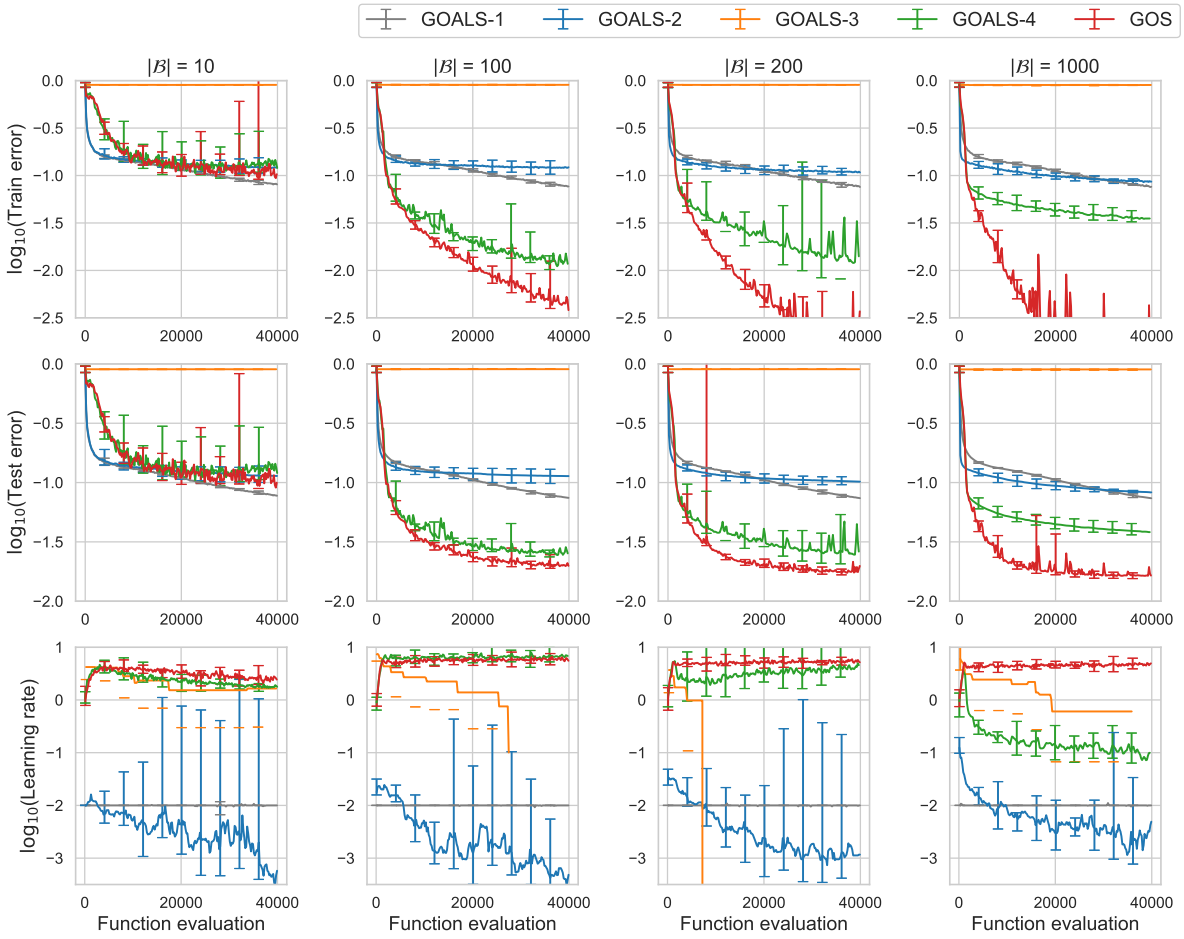


Figure 3.7: N-II MNIST dataset with batch size, $|\mathcal{B}| = 10$, 100, 200 and 1000 from left to right for various hyperparameters settings of GOALS which are listed in Table 3.1. The comparison of training dataset error (the 1st row), test set error (the 2nd row) and learning rate (the 3rd row) on a log_{10} scale versus the number of function evaluations.

Table 3.6 shows the top average training and test accuracies for each hyperparameter setting and the differences, $\psi_{y,|\mathcal{B}|}$, for each batch size on N-II. The ratios of test to training accuracies are given in the last column. Note that there are ratios greater than one. This might happen due to biases existing in the test dataset when the problem is simple to generalize. The lowest ratios are given by GOALS-3 except for the batch size, $|\mathcal{B}|$, of 100, meaning that GOALS-3 has

the least generalizability. Except when the batch size, $|\mathcal{B}|$, is 10, the difference values, $\psi_{y,|\mathcal{B}|}$, show that the aggressive vanilla GOS tends to outperform all GOALS settings again because no convergence condition restricts its learning rate.

Batch size, $\|\mathcal{B}\|$	Strategies	Train acc.	Diff., $\psi_{y,\|\mathcal{B}\|}$	Test acc.	Diff., $\psi_{y,\|\mathcal{B}\|}$	Te./Tr.
10	GOALS-1	**91.95**	0	**92.26**	0	1.003
	GOALS-2	88.05	3.9	88.63	3.63	1.007
	GOALS-3	10.05	81.9	10.02	82.24	0.997
	GOALS-4	89.4	2.55	89.55	2.71	1.002
	GOS	90.62	1.33	90.93	1.33	1.003
100	GOALS-1	92.33	7.29	92.57	5.49	1.003
	GOALS-2	88.05	11.57	88.66	9.4	1.007
	GOALS-3	9.95	89.67	9.93	88.13	0.998
	GOALS-4	98.89	0.73	97.55	0.51	0.986
	GOS	**99.62**	0	**98.06**	0	0.984
200	GOALS-1	92.43	7.44	92.62	5.64	1.002
	GOALS-2	89.23	10.64	89.83	8.43	1.007
	GOALS-3	9.98	89.89	9.78	88.48	0.98
	GOALS-4	98.8	1.07	97.53	0.73	0.987
	GOS	**99.87**	0	**98.26**	0	0.984
1000	GOALS-1	92.44	7.49	92.62	5.75	1.002
	GOALS-2	91.49	8.44	91.74	6.63	1.003
	GOALS-3	10.31	89.62	10.12	88.25	0.982
	GOALS-4	96.51	3.42	96.18	2.19	0.997
	GOS	**99.93**	0	**98.37**	0	0.984

Table 3.6: Top average training and test accuracies over the ten runs for the various GOALS settings, GOALS-1, GOALS-2, GOALS-3, GOALS-4, and GOS on the N-II architecture with different batch sizes, $|\mathcal{B}| = 10$, 100, 200, 1000 for the SGD optimizer. It also measures the difference, $\psi_{y,|\mathcal{B}|}$, between the performance and the best one from the different batch sizes. The ratios of test to training accuracies are given in the last column. The highest train and test accuracies are indicated in bold.

Table 3.7 shows the training and test relative robustness, R_y, and the average ratios of test to training accuracies of the results in Table 3.6 over the batch sizes. The measures indicate that GOS is the most robust strategy in both training and test results, followed by GOALS-4. Note that this result is consistent with the previous numerical study. Since the N-II architecture is not challenging to generalize, the average ratios are practically alike, but the results show that the least generalizable strategies are GOS and GOALS-3 on this problem with the smallest average ratios. Based on these results, we chose GOALS-4 for the additional comparison against other strategies.

Strategies	Training R_y	Test R_y	Avg. Te./Tr.
GOALS-1	22.22	16.88	1.003
GOALS-2	34.55	28.09	1.006
GOALS-3	351.08	347.1	0.989
GOALS-4	7.77	6.14	0.993
GOS	**1.33**	**1.33**	0.989

Table 3.7: The training and test relative robustness, R_y, for different strategies are given by summing the differences, $\Psi_{y,|\mathcal{B}|}$, given in Table 3.6. The average ratios of training to test accuracies for each strategy is given as the average values of the ratios listed in Table 3.6 across the different batch sizes. The lowest robustness measures are indicated in bold.

Performance comparison for different strategies

Next, Figure 3.8 shows the training error, test error, and learning rates for the ten strategies on the N-II architecture with MNIST. The constant learning rates generally show low variance

in error during training. Its error noticeably reduces from $|\mathcal{B}| = 10$ to $|\mathcal{B}| = 100$ since the SGD direction becomes more representative of the exact (full-batch) SGD direction. The learning rates are independent of noisy information when $|\mathcal{B}|$ is small. Hence, the more straightforward strategies may perform better than more sophisticated ones.

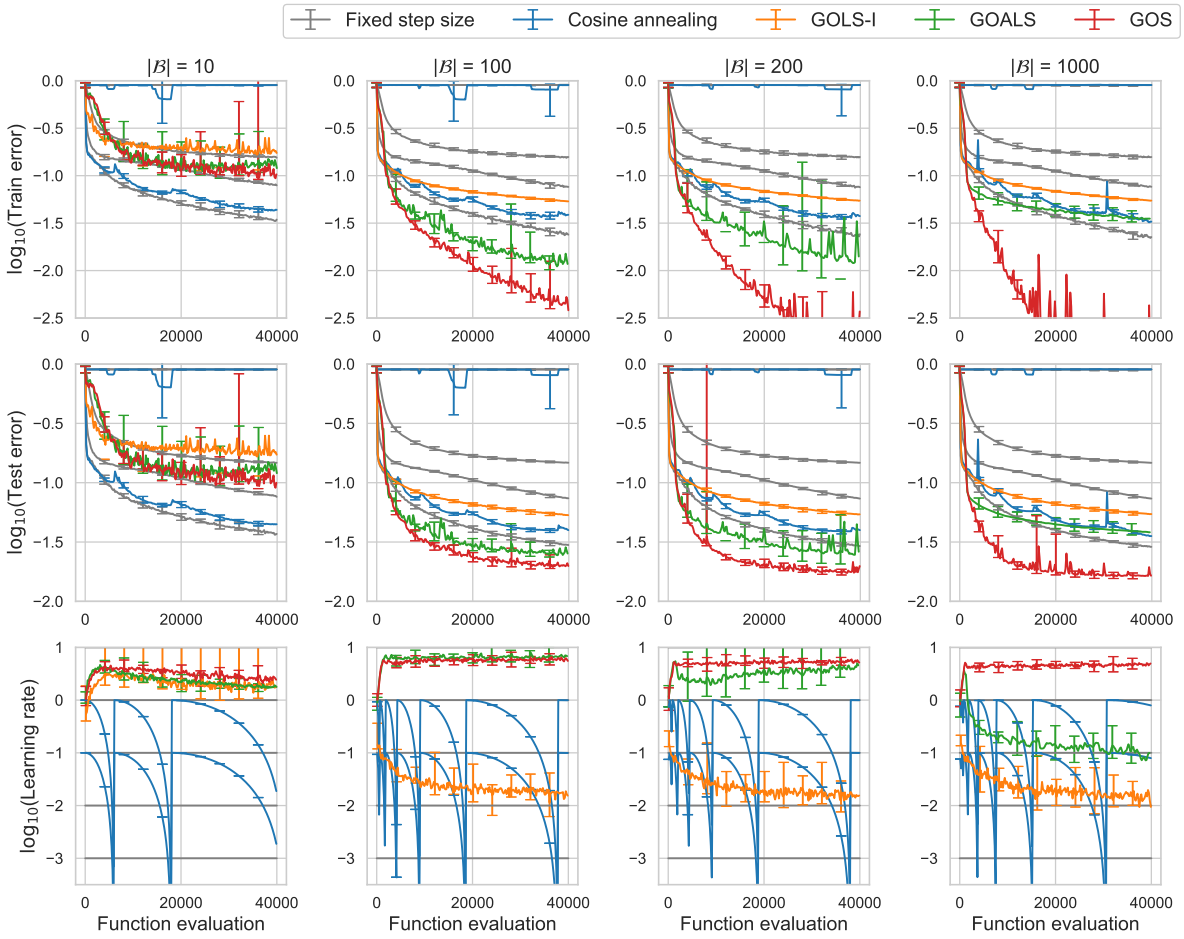


Figure 3.8: N-II MNIST dataset with batch size, $|\mathcal{B}| = 10$, 100, 200 and 1000 from left to right for various line search methods: constant learning rates, cosine annealing, GOLS-I, vanilla GOS, and GOALS-4. The comparison of training dataset error (the 1st row), test set error (the 2nd row) and learning rate (the 3rd row) on a $\log_{10}$ scale versus the number of function evaluations.

The best performing strategy for SGD overall is GOS, which is consistent with the observation in Section 3.6. Although the learning rates of GOS are continuously larger than others, SGD benefits from overshooting because it causes conjugacy in search directions. The training performance of GOS noticeably improves as $|\mathcal{B}|$ increases.

Cosine annealing with warm restarts changes the learning rates over periods of epoch and resets them to the initial learning rates. Hence, we observe slight fluctuations in both training and test errors. Although this method sweeps through an extensive range of learning rates, the results show that initial learning rate choice significantly affects the strategy's performance.

Both GOALS-4 and GOLS-I try to locate SNN-GPPs. However, GOALS-4, which uses the quadratic model, outperforms GOLS-I without any approximation model. Both strategies double the learning rates to grow, but GOALS-4 uses quadratic approximations to shrink, while GOLS-I halves the learning rates to shrink. Hence, GOALS-4 may perform worse if the quadratic approximation has a large approximation error due to noisy information.

Table 3.8 shows the top average training and test accuracies of each strategy and the differences measured for the same experiment. The last column in the table shows the ratios of test to training accuracies. The fixed learning rate results indicate that the performance of SGD differs considerably with the choice of learning rates. The fixed learning rate of 0.1 presents

the best training and test accuracies for the batch size of 10. For the larger batch sizes, GOS shows the highest accuracies of all strategies with SGD. The average ratios show that some are greater than one. This could mean the test data is biased when the problem is not challenging to generalize. The highest and lowest ratios are shown by the fixed learning rate of 0.001 and 10, respectively. When the ratios are lower, the strategy is more overfitted.

Batch size, $\|\mathcal{B}\|$	Strategies	Train acc.	Diff., $\psi_{y,\|\mathcal{B}\|}$	Test acc.	Diff., $\psi_{y,\|\mathcal{B}\|}$	Te./Tr.
10	Fixed, $\alpha = 0.001$	84.54	12.13	85.3	11.06	1.009
	Fixed, $\alpha = 0.01$	92.1	4.57	92.4	3.96	1.003
	Fixed, $\alpha = 0.1$	**96.67**	0	**96.36**	0	0.997
	Fixed, $\alpha = 1$	10.09	86.58	10.14	86.22	1.005
	Fixed, $\alpha = 10$	10.05	86.62	9.74	86.62	0.969
	Cosine, $\alpha = 0.1$	95.73	0.94	95.55	0.81	0.998
	Cosine, $\alpha = 1$	36.2	60.47	36.4	59.96	1.006
	GOLS-I	83.29	13.38	83.39	12.97	1.001
	GOALS-4	89.4	7.27	89.55	6.81	1.002
	GOS	90.62	6.05	90.93	5.43	1.003
100	Fixed, $\alpha = 0.001$	84.59	15.03	85.27	12.79	1.008
	Fixed, $\alpha = 0.01$	92.41	7.21	92.61	5.45	1.002
	Fixed, $\alpha = 0.1$	97.64	1.98	97.02	1.04	0.994
	Fixed, $\alpha = 1$	10.05	89.57	9.95	88.11	0.99
	Fixed, $\alpha = 10$	10.03	89.59	9.86	88.2	0.983
	Cosine, $\alpha = 0.1$	96.35	3.27	96.03	2.03	0.997
	Cosine, $\alpha = 1$	36.68	62.94	36.84	61.22	1.004
	GOLS-I	94.66	4.96	94.68	3.38	1
	GOALS-4	98.89	0.73	97.55	0.51	0.986
	GOS	**99.62**	0	**98.06**	0	0.984
200	Fixed, $\alpha = 0.001$	84.46	15.41	85.28	12.98	1.01
	Fixed, $\alpha = 0.01$	92.53	7.34	92.65	5.61	1.001
	Fixed, $\alpha = 0.1$	97.72	2.15	97.08	1.18	0.993
	Fixed, $\alpha = 1$	10.07	89.8	9.89	88.37	0.982
	Fixed, $\alpha = 10$	10.02	89.85	9.84	88.42	0.982
	Cosine, $\alpha = 0.1$	96.42	3.45	96.11	2.15	0.997
	Cosine, $\alpha = 1$	19.14	80.73	19.07	79.19	0.996
	GOLS-I	94.56	5.31	94.6	3.66	1
	GOALS-4	98.8	1.07	97.53	0.73	0.987
	GOS	**99.87**	0	**98.26**	0	0.984
1000	Fixed, $\alpha = 0.001$	84.42	15.51	85.29	13.08	1.01
	Fixed, $\alpha = 0.01$	92.39	7.54	92.65	5.72	1.003
	Fixed, $\alpha = 0.1$	97.79	2.14	97.12	1.25	0.993
	Fixed, $\alpha = 1$	10.31	89.62	10.14	88.23	0.984
	Fixed, $\alpha = 10$	10.06	89.87	9.94	88.43	0.988
	Cosine, $\alpha = 0.1$	96.77	3.16	96.45	1.92	0.997
	Cosine, $\alpha = 1$	18.5	81.43	18.52	79.85	1.001
	GOLS-I	94.55	5.38	94.57	3.8	1
	GOALS-4	96.51	3.42	96.18	2.19	0.997
	GOS	**99.93**	0	**98.37**	0	0.984

Table 3.8: Top average training and test accuracies over the ten runs for the SGD optimizer with various learning rate strategies, y, including the fixed learning rates, cosine annealing with warm restart, GOLS-I, GOALS-4, and GOS on the N-II architecture with different batch sizes, $|\mathcal{B}| = 10$, 100, 200, 1000. It measures the difference, $\psi_{y,|\mathcal{B}|}$, between the performance and the best one from the different batch sizes. The ratios of test to training accuracies are given in the last column. The highest train and test accuracies are indicated in bold.

Table 3.9 shows the training and test relative robustness, R_y, and the average ratios of test to training accuracies of the results in Table 3.8 over the different batch sizes. The most robust training and test results are obtained from GOS and the fixed learning rate of 0.1, respectively. GOALS-4 ranked fourth in both training and test results, led by GOS, the fixed learning rate of 0.1 and cosine annealing with $\alpha = 0.1$. However, note that the results of learning rate and cosine annealing are highly responsive to learning rates.

The table also computes the relative robustness measures and average ratios of each strategy when the impractical mini-batch size of 10 is excluded. Note that GOALS-4 now ranks third and second in both training and test robustness, respectively. This happens because the information for the batch size of 10 makes the information sparse, affecting the approximation accuracy for GOALS-4. The fixed learning rates and cosine annealing are not sensitive to the choice of the sparsity of the information like GOALS-4, but GOALS-4 can adopt learning rates based on the available information. The average ratio of 0.99 for GOALS-4 indicates that the quadratic function approximation has competitive generalizability. The average ratio of 0.99 for GOALS-4 indicates that the quadratic function approximation has competitive generalizability.

Strategies	Training R_y	Test R_y	Avg. Te./Tr.
Fixed, $\alpha = 0.001$	58.08 (30.54)	49.91 (38.85)	1.009 (1.009)
Fixed, $\alpha = 0.01$	26.66 (14.75)	20.74 (16.78)	1.002 (1.002)
Fixed, $\alpha = 0.1$	6.27 (4.12)	**3.47** (3.47)	0.994 (0.993)
Fixed, $\alpha = 1$	355.57 (179.19)	350.93 (264.71)	0.99 (0.985)
Fixed, $\alpha = 10$	355.93 (179.46)	351.67 (265.05)	0.981 (0.984)
Cosine, $\alpha = 0.1$	10.82 (6.43)	6.91 (6.1)	0.997 (0.997)
Cosine, $\alpha = 1$	285.57 (144.37)	280.22 (220.26)	1.002 (1)
GOLS-I	29.03 (10.34)	23.81 (10.84)	1(1)
GOALS-4	12.49 (4.15)	10.24 (3.43)	0.993 (0.99)
GOS	**6.05** (0)	5.43 (0)	0.989 (0.984)

Table 3.9: The training and test relative robustness, R_y, for different strategies are given by summing the differences, $\Psi_{y,|\mathcal{B}|}$, given in Table 3.8. The average ratios of training to test accuracies for each strategy are given as the average values of the ratios listed in Table 3.8 across the different batch sizes. The lowest robustness measures are indicated in bold. Additionally, the relative robustness, R_y, the average ratios measured excluding $|\mathcal{B}| = 10$ are listed in brackets.

3.8 Conclusions

Dynamic mini-batch sub-sampling (MBSS) in deep neural network problems causes the loss functions point-wise discontinuous. This makes the function value minimization approach impractical as line searches because it would find infinitely many local minima in discontinuous settings. Dynamic MBSS also causes sampling errors, manifesting as small bias and large variance. As a solution to minimize the variance, a recent study introduced gradient-only surrogates (GOS) to resolve learning rates. GOS is a quadratic function approximation model constructed using only directional derivative information. It approximates the spatial locations of sign changes in directional derivatives to choose learning rates. The previous study showed the competitive performance of GOS in training DNNs. However, it does not have a convergence proof developed, and it makes GOS not conservative.

Hence, this study extends the vanilla GOS to be conservative by limiting GOS to be convergent using a bracketing strategy, with provided proof. We implemented the Wolfe curvature condition as the convergence proof because it only requires directional derivative information. The bracketing strategy is empowered by the Regular-Falsi method. It aims to restrict the domain of GOS for higher model accuracy because the accuracy could have been reduced by implementing the simplistic quadratic models for computational efficiency. Unlike GOS, which constructs an approximation once for a descent direction, GOALS consecutively constructs the GOS models until the curvature condition is satisfied. This makes GOALS more robust in terms of convergence.

We introduced a new relative robustness measure for assessing the performance of GOALS. The traditional performance measure for a learning rate strategy only considers the top performance for a specific problem and optimizer. On the other hand, our relative robustness measure considers all accounts, including poor performances. Hence, the robustness measure favors strategies that perform well over different optimizers and problems.

We conducted hyperparameter studies for GOALS on ResNet-18 and EfficieintNet-B0 with the CIFAR-10 dataset using various optimizers, including SGD, RMSProp, and Adam. Testing on various optimizers showed the adaptability of GOALS as a learning rate strategy. Based on the robustness measure, we learned it is essential to choose the data point close to the origin to reduce the model error. Using the fixed recommended learning rate for the specific optimizer to determine the data point for every descent direction is turned out to be a better choice to reduce the approximation error than using the previously resolved learning rate as the next initial guess. The experimental results showed that one hyperparameter setting close to GOS outperformed GOS in both training and test accuracy for both test problems. It also showed that GOALS, led by GOS, generalizes better and is more robust than the recommended learning rates for each optimizer.

We further compared the performance of GOALS against nine other learning rate strategies on a shallower DNN, N-II, using only the SGD optimizer, as the performance of SGD is sensitive to learning rates. For this less nonlinear problem compared to ResNet-18 and EfficientNet-B0, extending the initial data point for GOALS farther than the recommended learning rates helped with performances based on the relative robustness measures. On the N-II architecture, GOALS ranked third and second for the training and test robustness based on the relative robustness measures, respectively, led by less conservative GOS among ten learning rate strategies in total. The results show that quadratic approximations warrant further investigation.

Chapter 4

GOCLS: Gradient-Only Line Search With Bayesian Classification Approach For Training Neural Networks

4.1 Chapter overview

The learning rate is a critical hyperparameter that requires tuning in deep neural network (DNN) training. Strategies to resolve learning rates can be classified as follows: 1) fixed learning rates, 2) decay methods, 3) cyclical methods, and 4) line searches. Along a descent direction, line searches can be designed to find a minimizer point, an optimality criterion point (necessary condition), or a non-negative gradient projection point (NN-GPP). However, these line searches either approximate the loss function or directly optimizes the loss function. Both may be computationally demanding. This paper investigates the potential of classification as an alternative paradigm to resolve learning rates. This is enabled through gradient-only optimization principles, where we can resolve an NN-GPP by finding a sign change from negative to positive. The proposed approach is adaptive and based on Bayesian classification that uses historical directional derivative information to discriminate between negative and positive signs. This only requires one gradient computation per line search iteration. In addition, the line search hyperparameter is interpretable. It implies the probability of seeing a positive or negative sign for the directional derivative. The approach is adaptable to optimizers such as SGD, RMSProp, and Adam. This study compares our proposed classification-based line search to other learning rate strategies that include constant learning rate, step decay, cosine annealing, and different gradient-only line searches that include GOS and GOALS based on a newly proposed robustness measure. Instead of considering only the best performance, we consider the reliability of an algorithm to perform well across optimizers and problems. The architectures we consider are ResNet-18 and EfficientNet-B0 on the CIFAR-10 dataset.

4.2 Introduction

A learning rate is a hyperparameter that is critical for successful deep neural network training. The learning rate is dependent on various factors such as optimizers, models, datasets, and preprocessing of the data. Hence, there are many popular learning rate strategies: fixed or constant learning rates, learning rate decay methods, cyclic learning rates, and line searches as listed in Table 4.1. Conventional line search methods attempt to search for a learning rate to locate a minimizer point that satisfies its desired optimality criterion, often minimizing the function value or locating where directional derivatives' sign changes from negative to positive, so-called non-negative gradient projection point (NN-GPP). This is usually accomplished by

either consecutively evaluating the functions or indirectly constructing approximation models. Several attempts to apply line search methods for training deep neural networks (DNN) in a stochastic setting have been attempted. They are the probabilistic line search using Bayesian optimization [Mahsereci and Hennig, 2015], the line search for locating Stochastic NN-GPP (SNN-GPP) with [Chae and Wilke, 2019a, Chae et al., 2021] or without [Kafka and Wilke, 2019a] using approximation models.

Class	Examples	Required information	Citation
Constant	Fixed learning rate	None	-
Decay	Step	Epoch	[Paszke et al., 2019]
	Exponential	Epoch	[Paszke et al., 2019]
Cyclic	Cosine annealing	Epoch	[Loshchilov and Hutter, 2016]
	Cyclic	Epoch	[Smith, 2017]
Line search	Probabilistic	Loss and gradient	[Mahsereci and Hennig, 2015]
	Gradient-only	Gradient	[Kafka and Wilke, 2019a, Chae and Wilke, 2019a] [Chae et al., 2021]
Classification	GOCLS	Gradient	-

Table 4.1: Various classes and examples of learning rate strategies and their required information.

In this study, we introduce a new class to adaptively determine learning rates using a classification. This is enabled by gradient-only optimization principles [Snyman and Wilke, 2018], particularly, that of locating a non-negative gradient projection point (NN-GPP). An NN-GPP along a descent direction manifest as a sign change from negative to positive. This naturally enables classification to be considered as a paradigm to find learning rates. It only requires historic spatial locations of negative and positive signs of directional derivatives along the descent search direction. Hence, the gradient computation per iteration is limited to one. Our line search algorithm is similar to one dimensional Linear Discriminant Analysis (LDA) [Friedman et al., 2001]. We use Bayesian inference to find a location that separates the negative directional derivative signs from the positive. An NN-GPP along a descent direction is assumed to have a 50% chance of observing positive or negative directional derivatives for different mini-batches. The advantage of having such an interpretable hyperparameter is that we can choose whether we want to resolve, overshoot or undershoot an NN-GPP.

Note that the adaptive learning rate strategies such as Adam [Kingma and Ba, 2014], Adadelta [Zeiler, 2012b], RMSProp [Tieleman and Hinton, 2012] are not included in Table 4.1. This is because we consider optimizers for which the search direction is coupled with the directional learning rates. Using these optimizers, we compared various learning rate strategies: 1) a decay learning rate, Step Decay, 2) a cyclic learning rate [Smith, 2017], Cosine Annealing [Loshchilov and Hutter, 2016], 3) constant learning rate, 4) GOS [Chae and Wilke, 2019a], 5) GOALS on ResNet-18 [He et al., 2016] and EfficientNet-B0 [Tan and Le, 2019] for the CIFAR-10 dataset [Krizhevsky, 2009].

Current researches in this field mainly consider only the best performances to rank different strategies. This only allows us to detect the strategies that are specialized for specific architectures. Because neural network architectures keep growing in size and more complex to train every year [Brown et al., 2020], we need to distinguish strategies that work well across different architectures to create a benchmark result. We managed this by considering all accounts, including poor performances, for comparison since this allows us to quantify the least bad performance on average. We formally define the relative robustness measure in the paper. The results showed that our classification-based strategy is competitive against six other strategies based on the measure.

4.3 Related work

This section briefly explains the dynamic mini-batch sub-sampling (MBSS), which we implement throughout the research for training DNN problems, and discusses the characteristics of various

types of learning rate strategies available. Last, we introduce a new concept of predicting the location of minima using a classification approach, leading to the derivation of our line search algorithm in Section 4.4.

4.3.1 Dynamic mini-batch sub-sampling

Mini-batch sub-sampling is essential in deep learning because it resolves the issues related to the limited amount of computational memory and helps with regularization by focusing on a small bit of information in train data. We introduce the notations for the full-batch evaluation and move on to the dynamic mini-batch sub-sampling, which we implement throughout the paper. Whereas a neural network full-batch loss function, $\mathcal{L}(\boldsymbol{x})$, of the weight, $\boldsymbol{x}$, is given by

$$\mathcal{L}(\boldsymbol{x}) := \frac{1}{M}\sum_{b=1}^{M} \ell(\boldsymbol{x}; \boldsymbol{t}_b), \tag{4.1}$$

a mini-batch sub-sample (MBSS) loss, $L(\boldsymbol{x})$, is given by

$$L(\boldsymbol{x}) := \frac{1}{|\mathcal{B}|}\sum_{b\in\mathcal{B}} \ell(\boldsymbol{x}; \boldsymbol{t}_b), \tag{4.2}$$

where the individual sample loss, $\ell(\boldsymbol{x}; \boldsymbol{t}_b)$, is computed using each training sample, $\boldsymbol{t}_b \in \mathbb{R}^{D\times 1}$, taken from the training sample set, $\{\boldsymbol{t}_1, \boldsymbol{t}_2, ..., \boldsymbol{t}_M\}$. The mini-batch subset, $\mathcal{B} \subset \{1, 2, 3, ..., M\}$, with size of $|\mathcal{B}| \ll M$ is uniform randomly sampled for computing, $\ell(\boldsymbol{x}; \boldsymbol{t}_b)$. In addition, the gradient of the full-batch loss function, $\boldsymbol{\nabla}\boldsymbol{\mathcal{L}}(\boldsymbol{x})$, and the gradient of MBSS loss function, $\boldsymbol{g}(\boldsymbol{x})$, are given by

$$\boldsymbol{\nabla}\boldsymbol{\mathcal{L}}(\boldsymbol{x}) := \frac{1}{M}\sum_{b=1}^{M} \boldsymbol{\nabla}\boldsymbol{\ell}(\boldsymbol{x}; \boldsymbol{t}_b), \tag{4.3}$$

and

$$\boldsymbol{g}(\boldsymbol{x}) := \frac{1}{|\mathcal{B}|}\sum_{b\in\mathcal{B}} \boldsymbol{\nabla}\boldsymbol{\ell}(\boldsymbol{x}; \boldsymbol{t}_b), \tag{4.4}$$

respectively, where $\boldsymbol{\nabla}\boldsymbol{\ell}(\boldsymbol{x}; \boldsymbol{t}_b) \in \mathbb{R}^{p\times 1}$ is the individual sample gradient vector evaluated using a single sample, $\boldsymbol{t}_b$.

In the context of line search, we may formulate the MBSS loss function using either static MBSS, $\bar{L}(\boldsymbol{x})$, or dynamic MBSS, $\tilde{L}(\boldsymbol{x})$. A dynamic MBSS loss function, $\tilde{L}(\boldsymbol{x})$, is given by

$$\tilde{L}(\boldsymbol{x}) := \frac{1}{|\mathcal{B}_{n,i}|}\sum_{b\in\mathcal{B}_{n,i}} \ell(\boldsymbol{x}; \boldsymbol{t}_b), \tag{4.5}$$

where $\mathcal{B}_{n,i}$ denotes the mini-batch re-sampled at the i-th function evaluation (either loss value or gradient evaluation) of the n-th iteration. As a result, the dynamic MBSS loss function, $\tilde{L}(\boldsymbol{x})$, is point-wise discontinuous with small biases but large variances in the solutions of each loss functions. In contrast, the static MBSS loss function, $\bar{L}(\boldsymbol{x})$, is continuous. But still, there are large biases and small variances in the solutions of each loss function.

4.3.2 Comparisons of learning rate strategies

As the learning rate is a critical hyperparameter to be adjusted appropriately for successful neural network problems training, researchers often employ numerous learning rate strategies. We broadly categorize them into four groups: 1) fixed learning rates, 2) decay methods, 3) cyclic methods, and 4) line searches.

Since the fixed learning rates are monotonous, it requires the user to optimize the learning rate for the problem. The decay methods start with relatively larger learning rates for practical exploration and reduce the learning rates for exploitation close to training. The cyclic methods

oscillate the learning rates at a specified rate. Both methods are effective for improving the convergence rates. However, it might not be obvious to select the appropriate hyperparameters without prior knowledge about the problems.

Line searches are adaptive to the various problem settings and environments, which requires minimum prior knowledge about the problems. However, it often requires more than one function evaluation for a descent direction to search for learning rates that satisfy optimality criteria discussed in the previous section.

Lastly, the line searches utilize optimization techniques to search for learning rates that satisfy optimality criteria along a descent direction. Although this method is adaptive to various problem settings and requires minimum prior knowledge about the problems, it often requires more than one function evaluation per iteration. In the next section, we study two distinct objective functions for line searches. One minimizes the function values, and the other locates where the signs of the directional derivatives change.

4.3.3 Objective functions for optimization approaches

Before studying the two objective functions in this section, we want to clarify that both static and dynamic MBSS work identically when an optimizer evaluates the DNN function just once every iteration or search direction. The difference only occurs when multiple function evaluations are required at each iteration, such as line searches.

Function value minimization

For a smooth function such as a static MBSS loss function, $\bar{L}(\boldsymbol{x})$, we implement minimization formulation along a chosen search direction, $\boldsymbol{d}_n$, given by

$$\arg\min_{\alpha} \bar{L}_n(\boldsymbol{x} + \alpha \boldsymbol{d}_n), \tag{4.6}$$

where α denotes the learning rate. However, for a point-wise discontinuous function such as a dynamic MBSS loss function, $\tilde{L}(\boldsymbol{x})$, we need an alternative way to formulate the objective function since (4.6) finds irrelevant multiple local minima.

Stochastic non-negative gradient projection point

As a solution to the previous matter, we could alternatively search for stochastic non-negative gradient projection points (SNN-GPPs) which is given by

$$\boldsymbol{d}_n^{\intercal} \tilde{\boldsymbol{g}}(\boldsymbol{x}_{snngpp} + \alpha_n \boldsymbol{d}_n) \geq 0, \;\; \forall \|\boldsymbol{d}_n \in \mathbb{R}^p\|_2 = 1, \;\; \forall \alpha \in (0, \alpha_{max}], \;\; p(\boldsymbol{x}_{snngpp}) > 0, \tag{4.7}$$

with probability, $p(\boldsymbol{x}_{snngpp})$, greater than 0. SNN-GPPs expect to observe positive directional derivatives when moving away from the points by a given learning rate, α, along the search direction, $\boldsymbol{d}_n$. It implies that SNN-GPPs are the possibly local minima in a stochastic discontinuous function. Kafka and Wilke [2019a] extended the definition of SNN-GPP from NN-GPP [Wilke et al., 2013] given by this,

$$\boldsymbol{d}_n^{\intercal} \boldsymbol{\nabla} \boldsymbol{\mathcal{L}}(\boldsymbol{x}_{nngpp} + \alpha_n \boldsymbol{d}_n) \geq 0, \quad \forall \|\boldsymbol{d}_n \in \mathbb{R}^p\|_2 = 1, \quad \forall \alpha \in (0, \alpha_{max}], \tag{4.8}$$

which only accommodates a deterministic discontinuous function. The following section introduces our line search algorithm, which aims to locate the SNN-GPPs using only gradient information with a Bayesian statistics perspective.

4.3.4 Comparisons of line searches for training DNNs

We may categorize various line search algorithms by different MBSS techniques, approximation model usages, optimality criteria, and information they utilize. [Mutschler and Zell, 2019] uses a deterministic parabolic approximation built with function value and gradient information to search for local minima in the static MBSS setting. In dynamic MBSS settings, Chae and Wilke [2019a] and Chae et al. [2021] approximate the location of sign changes using a deterministic quadratic approximation model built only with gradient information in the dynamic MBSS setting. [Mahsereci and Hennig, 2015] utilizes a stochastic Bayesian optimization model with both function value and gradient information to locate local minima. [Kafka and Wilke, 2019a] deterministically searches for SNN-GPPs only with gradient information without approximation models.

4.3.5 Relative robustness measure, R

As DNN problems have become more complex, we might prefer using a robust optimizer that performs well across various problems to a problem-specific optimizer for training an unseen problem. In other words, we would be interested in the strategy not necessarily outperforms all the other optimizers for only a specific problem but has the slightest differences in performance from the best version of each problem. Hence, we define the relative robustness measure [Chae et al., 2021] for learning rate strategies as follows:

Definition 4.3.1 (Relative robustness)**.** *Given a line search strategy, y, optimizer, o, and problem, h, the absolute performance differences, $\psi_{y,h,o}$, is computed as the difference between the strategy's accuracy, $\eta_{y,h,o}$, and the overall best accuracy, $\eta^*_{h,o}$. The relative robustness, R_y, of a strategy, y, is computed by summing the absolute performance difference over all optimizers and all problems given by*

$$R_y = \sum_{h \in H} \sum_{o \in O} \psi_{y,h,o}, \text{ where } \psi_{y,h,o} = |\eta^*_{h,o} - \eta_{y,h,o}|. \tag{4.9}$$

Hence, the lower the measure, R_y, the more robust the strategy. One may also compute a robustness measure, $R_{y,h}$, for a strategy, y, and a specific problem, h, while considering all optimizers, O. We will compare the training and test performance of our proposed algorithm, GOALS, against the other learning rate strategies based on the relative robustness measure throughout the paper.

4.4 Gradient-only classification line search

This section introduces the gradient-only classification line search (GOCLS), which combines the ideas of Bayesian probability and the observation of sign-changes to build a classification-based line search algorithm. As we call the cluster of SNN-GPPs, the ball, B, we assume that, at any point inside B along the given search direction, each probability of observing the positive or negative signs of directional derivatives are between 0 and 1, while they sum up to 1.

Assuming that the properties of B in the current iteration are closely related to the ones observed iterations in the past, we predict the learning rates that are likely to provide the target probability of observing positive or negative signs. In other words, it allows controlling whether we need to overshoot or undershoot while remaining inside the ball. We later show in the experiment that this parameter considerably influences the performance in training.

GOCLS focuses on locating the SNN-GPPs from a Bayesian perspective. We try to predict the learning rate required to get to the local minima using the historical data accumulated during the training. Hence, we assumed that at the local minima, the probability of observing positive signs, π_ξ, is about the same as observing negative signs, π_ζ, throughout different mini-batches over iterations. Note that we distinguish between π_s and $\hat{\pi}_s$ as the target probability and the sample probability in this paper, respectively.

We follow linear discriminant analysis (LDA) [Friedman et al., 2001] to locate such points since we assume that the variance of the probability density function (pdf) of the positive signs is identical to that of the negative signs. The difference is that we use a single random variable which is the learning rate, α, instead of using a bivariate Gaussian distribution. This assumption is appropriate based on the single-mode assumption when we scale the learning rates on the $\log_{10}$ scale.

4.4.1 Derivation of gradient-only classification line search (GOCLS)

We start the derivation of GOCLS by matching the posterior probabilities of observing a sign class, S, that is positive, ξ and that is negative, ζ, given the observations of learning rates, α, for a target probability of observing the positive signs, π_ξ.

$$p(S = \xi | A = \alpha)\left(\frac{1 - \pi_\xi}{\pi_\xi}\right) = p(S = \zeta | A = \alpha), \tag{4.10}$$

where

$$p(S = s | A = \alpha) = \frac{q_s \hat{\pi}_s}{\sum_{k \in S} q_k \hat{\pi}_k}. \tag{4.11}$$

The prior, $p(S = s) = \hat{\pi}_s$, is the sample probability which is the ratio of the number of samples in class s to the total number of samples, $|N|$, in the sample set, N. The likelihood, $p(A = \alpha | S = s) = q_s(\alpha)$, is a pdf of the Gaussian distribution, given by

$$q_s(\alpha) = \frac{1}{\sigma_s \sqrt{2\pi}} e^{-\frac{1}{2}(\frac{\alpha - \mu_s}{\sigma_s})^2} \tag{4.12}$$

where μ_s and σ_s are the mean and the standard deviation of each class. However, as mentioned earlier, we assume that the standard deviation for the positive class, σ_ξ, and the negative class σ_ζ are identical to be the overall standard deviation, σ_a. Hence, (4.11) is rewritten after taking the log of both sides as follows:

$$\log(q_s(\alpha)\hat{\pi}_s) = -\log(\sigma_a \sqrt{2\pi}) - \frac{1}{2}\left(\frac{\alpha - \mu_s}{\sigma_a}\right)^2 + \log(\hat{\pi}_s) \tag{4.13}$$

$$= \frac{\alpha \mu_s}{\sigma_a^2} - \frac{\mu_s^2}{2\sigma_a^2} + \log(\hat{\pi}_s). \tag{4.14}$$

Therefore (4.10) can be simplified as follows:

$$\log\left(\frac{\hat{\pi}_\xi}{\hat{\pi}_\zeta}\right) + \log\left(\frac{1 - \pi_\xi}{\pi_\xi}\right) + \frac{\alpha(\mu_\xi - \mu_\zeta)}{\sigma_a^2} - \frac{\mu_\xi^2 - \mu_\zeta^2}{2\sigma_a^2} = 0 \tag{4.15}$$

Now solving for learning rate, α, that satisfies (4.10) is given by

$$\alpha = -\left[\log\left(\frac{\hat{\pi}_\xi}{\hat{\pi}_\zeta}\right) + \log\left(\frac{1 - \pi_\xi}{\pi_\xi}\right)\right]\frac{\sigma_a^2}{\mu_\xi - \mu_\zeta} + \frac{\mu_\xi + \mu_\zeta}{2}, \tag{4.16}$$

where the overall variance, σ_a^2, is given [Friedman et al., 2001] by

$$\sigma_a^2 = \frac{\sum^S \sum_{i=1}^N (\alpha^{(i)} - \mu_s)^2}{|N| - |S|}. \tag{4.17}$$

Note that $|N|$ and $|S|$ denote the cardinality of the learning rate sample set, N, and class set, S, respectively.

When we blindly accept the learning rate obtained from (4.16), the sample domain will shrink, and no more exploration occurs in training. In other words, it needs a balance between exploration and exploitation when resolving the learning rates. Hence, we take care of exploration introducing an exploration function in the form of a normal distribution, $\mathcal{N}(\mu_e, \sigma_e)$, with

$$\mu_e = \log_{10}(\|\boldsymbol{d}_n\|_2^{-1}) \text{ and } \sigma_e^2 = 1. \tag{4.18}$$

The exploration mean, μ_e, is the inverse of the L-2 norm of search direction on a $\log_{10}$ scale, and the variance, σ_e^2, is assumed to be 1. We compute the final learning rate on a $\log_{10}$ scale, α_f, by combing both exploitation and exploration normal distributions, $\mathcal{N}(\alpha, \sigma_a^2)$ and $\mathcal{N}(\mu_e, \sigma_e^2)$, respectively, in a closed-form as follows:

$$\alpha_f = \frac{\sigma_e^2}{\sigma_e^2 + \sigma_a^2}\alpha + \frac{\sigma_a^2}{\sigma_e^2 + \sigma_a^2}\mu_e \tag{4.19}$$

When employing the learning rate for training, we take the final learning rate on a $\log_{10}$ scale back to the standard scale,

$$\tilde{L}_{n+1}(\boldsymbol{x}) = \tilde{L}_n(\boldsymbol{x} + \tilde{\alpha}_n^* \boldsymbol{d}_n);\ \tilde{\alpha}_n^* = 10^{\alpha_f}. \tag{4.20}$$

4.4.2 Pseudo-code for GOCLS

In this section, we present the pseudo-code of GOCLS in Algorithm 5. It only requires a single function evaluation per iteration. Hence, it does not require any extra function evaluation when implemented for other optimizers for choosing the learning rate.

GOCLS requires several values as inputs for every iteration: the lists of learning rates, A_n, the list containing the directional derivative signs, S_n, observed for the past ω number of iteration, the search direction, $\boldsymbol{d}_n$, and the target probability of observing positive signs, π_ξ.

The algorithm creates two lists for learning rates resulting in positive directional derivative signs, $A_{\xi,n}$ and negative signs, $A_{\zeta,n}$. Once the lengths of both lists reach two, we begin to find the means, μ_ξ and μ_ζ, and the overall standard deviation, σ_a, for both classes, which leads to the computation of the exploitation learning rate, α in (4.16). Next, we find a balancing value, α_f, satisfying both the exploitation learning rate, α, and exploration mean, μ_e, based on their corresponding variances. We then rescale the computed learning rate in a $\log_{10}$ scale, α_f, back to the original scale and call it the final learning rate, $\tilde{\alpha}_n^*$. Lastly, the iteration ends with computing the new gradients, $\tilde{g}_n^*$, at the final learning rate, $\tilde{\alpha}_n^*$, storing the learning rate and the calculated gradient signs. We also ensure that the length of both the sign list, $|S_n|$, and learning rate list, $|A_n|$, does not exceed the prescribed length, ω. The simplified flowchart of GOCLS is shown in Figure 4.1.

4.5 Numerical study design

We conducted two numerical studies for the proposed line search algorithm, GOCLS. First, we conduct a hyperparameter study to search for appropriate target positive probabilities, π_ξ, for different optimizers, including SGD, RMSProp, and Adam. Learning rates selected using GOCLS are based on the choice of π_ξ. Hence, we aim to know which π_ξ produces the best performance for the different optimizers. Second, we conduct a performance comparison study of GOCLS using the previously chosen hyperparameters to compare the performance against the other strategies. We measure the performance of each strategy using the relative robustness measure introduced in Section 4.3.5. In the following sections, we discuss the details of the two numerical studies.

4.5.1 Numerical study 1: Hyperparameter studies

The first experiment aims to demonstrate the significance of the target probability parameter, π_ξ, by varying the parameter range from 40% to 90% with a 10% increment. It implies that we test undershooting by reducing it below 50% and overshooting by increasing it over 50%. This detailed study is conducted on ResNet-18 [He et al., 2016] for 350 epochs using the CIFAR-10 dataset [Krizhevsky, 2009] for the SGD, RMSProp and Adam optimizers. Hence, by no means

Algorithm 5: GOCLS

Input: list of learning rates, A_n, list of signs, S_n, search direction, $\boldsymbol{d}_n$, size of window, ω, target probability of observing positive signs, π_ξ

Output: resolved learning rate, $\tilde{\alpha}_n^*$, list of learning rates, A_{n+1}, list of signs, S_{n+1}

$A_{\xi,n} \rightarrow \{\alpha_i | s_i \geq 0, s_i \in S_n, \alpha_i \in A_n\}$
$A_{\zeta,n} \rightarrow \{\alpha_j | s_j < 0, s_j \in S_n, \alpha_j \in A_n\}$
$\mu_e \rightarrow \|\boldsymbol{d}_n\|_2^{-1}$
if $|A_{\xi,n}| \leq 1$ *and* $|A_{\zeta,n}| \leq 1$ **then**
  $\tilde{\alpha}_n^* \rightarrow \mu_e$
else
  $\hat{\pi}_\xi \rightarrow \frac{|A_{\xi,n}|}{|A_{\xi,n}|+|A_{\zeta,n}|}$
  Compute μ_ξ, μ_ζ
  $\sigma_a^2 \rightarrow \frac{\Sigma(A_{\xi,n}-\mu_\xi)^2+\Sigma(A_{\zeta,n}-\mu_\zeta)^2}{|A_{\xi,n}|+|A_{\zeta,n}|-2}$
  $\alpha \rightarrow \frac{\mu_\xi+\mu_\zeta}{2} - [\log(\frac{\hat{\pi}_\xi}{1-\hat{\pi}_\xi}) + \log(\frac{1-\pi_\xi}{\pi_\xi})](\frac{\sigma_a^2}{\mu_\xi-\mu_\zeta})$
  $\alpha_f \rightarrow \frac{\sigma_e^2}{\sigma_e^2+\sigma_a^2}\alpha + \frac{\sigma_a^2}{\sigma_e^2+\sigma_a^2}\mu_e$
  $\tilde{\alpha}_n^* \rightarrow 10^{\alpha_f}$
Compute $\tilde{g}_n^*$ at $\tilde{\alpha}_n^*$
$\tilde{f}_n' \rightarrow \boldsymbol{d}_n^\intercal \tilde{g}_n^*$
$S_{n+1} \rightarrow [S_n, sign(\tilde{f}_n')]$
$A_{n+1} \rightarrow [A_n, \log_{10}(\tilde{\alpha}_n^*)]$
if $|A_{n+1}| > \omega$ **then**
  $S_{n+1} \rightarrow S_{n+1}.pop(0)$
  $A_{n+1} \rightarrow A_{n+1}.pop(0)$

do we try to determine the optimum π_ξ covering all kinds of problems. Nonetheless, the results demonstrate that overshooting rather than undershooting outperforms creating conjugacy in the subsequent search direction, $\boldsymbol{d}_n$.

4.5.2 Numerical study 2: Performance comparison

The second experiment aims to compare the performance of GOCLS using one of the best performing hyperparameter values, π_ξ, from the previous experiment against the results of the popular learning rate strategies. These include the fixed learning rates, GOS [Chae and Wilke, 2019a], GOALS [Chae et al., 2021], step decay [Paszke et al., 2019], and cosine annealing [Loshchilov and Hutter, 2016]. For the second experiment, we use ResNet-18 [He et al., 2016] and EfficientNet-B0 [Tan and Le, 2019] for 350 epochs using the CIFAR-10 dataset for the SGD, RMSProp [Tieleman and Hinton, 2012], and Adam [Kingma and Ba, 2014] optimizers.

We set the fixed learning rates to be the same as the recommended ones for each optimizer. Those are 0.01 for SGD and RMSProp and 0.001 for Adam. The step decay uses the multiplicative factor of 0.1 for every 100 epoch starting from the recommended learning rate for each optimizer. For the cosine annealing, we prepared two maximum numbers of iterations, T_{max}: 50 and 350. For the GOALS strategy, we choose GOALS-4, the hyperparameter setting recommended by Chae et al. [2021] to train DNN problems. GOALS is a line search method that constructs quadratic approximations only using gradient information to locate the sign changes of directional derivatives from negative to positive.

To compare the performances of the learning rate strategies, we employ the relative robustness measure introduced in Section 4.3.5. The measure is computed by summing the differences between the training or test accuracies and the best strategy's accuracy for each tested optimizer. Additionally, we measure the ratios of the test to training accuracies of different strategies. This means that the closer the ratios are to one, the more minor the relative discrepancy between the

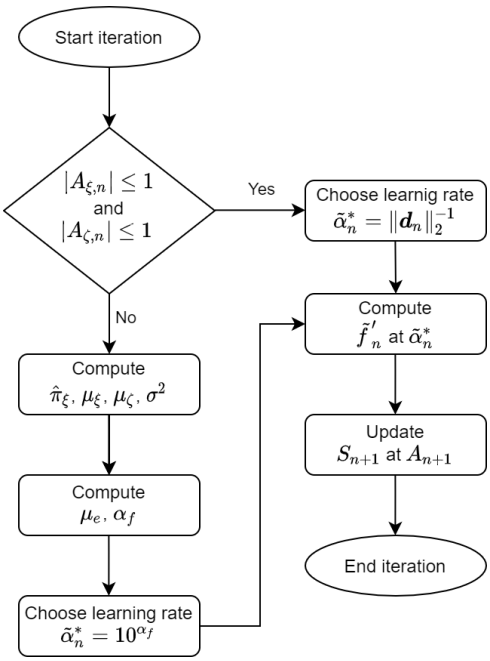


Figure 4.1: Simplified flowchart of the GOCLS algorithm

training and test accuracies is. The DNN generalizes well as it shows abilities to perform well on unobserved inputs [Bengio et al., 2017].

4.5.3 Standard experiment setting for numerical study 1 and 2

The observation window size, ω, chosen for both experiments is 100. We conducted the numerical experiments with ResNet-18 and EfficientNet-B0 using the PyTorch code set up by Liu [2020]. The batch size for the experiments is 128. We repeat each case in the experiments five times for 350 epoch to generate errorbar graphs. Note that since we adopt dynamic MBSS in the research, the epoch number is directly proportional to the number of gradient evaluations each strategy has taken. This means that when the average number of gradient evaluations per iteration is higher for a strategy, fewer descent directions are allowed to iterate. Since GOCLS evaluates gradient only once every iteration, it searches for minimum using the maximum number of descent directions allocated.

4.6 Results of numerical study

4.6.1 Numerical study 1: hyperparameter study

Choosing π_ξ for SGD

Figure 4.2 shows the training error, test error, learning rates on the $\log_{10}$ scale and the sample probability of positive signs, $\hat{\pi}_\xi$, along the 350 epoch for various π_ξ values. The train and test errors show that the more we allow overshooting with higher π_ξ, the lower errors become. Suppose we want to maintain the sample probability as high as the target probability. In that case, the learning rate must exponentially grow because it becomes challenging for the sample probability to match the desired one close to the end of training.

The lowest target probability, 40%, shows that the learning rate rapidly approaches the minimum bound for the learning rates. Although the highest target probability, 90%, shows the quickest convergence rate, it also indicates erratic training close to the end. We decided to select a target probability of 70% for SGD in the further experiments as it shows the lowest training error without instability as with 90%.

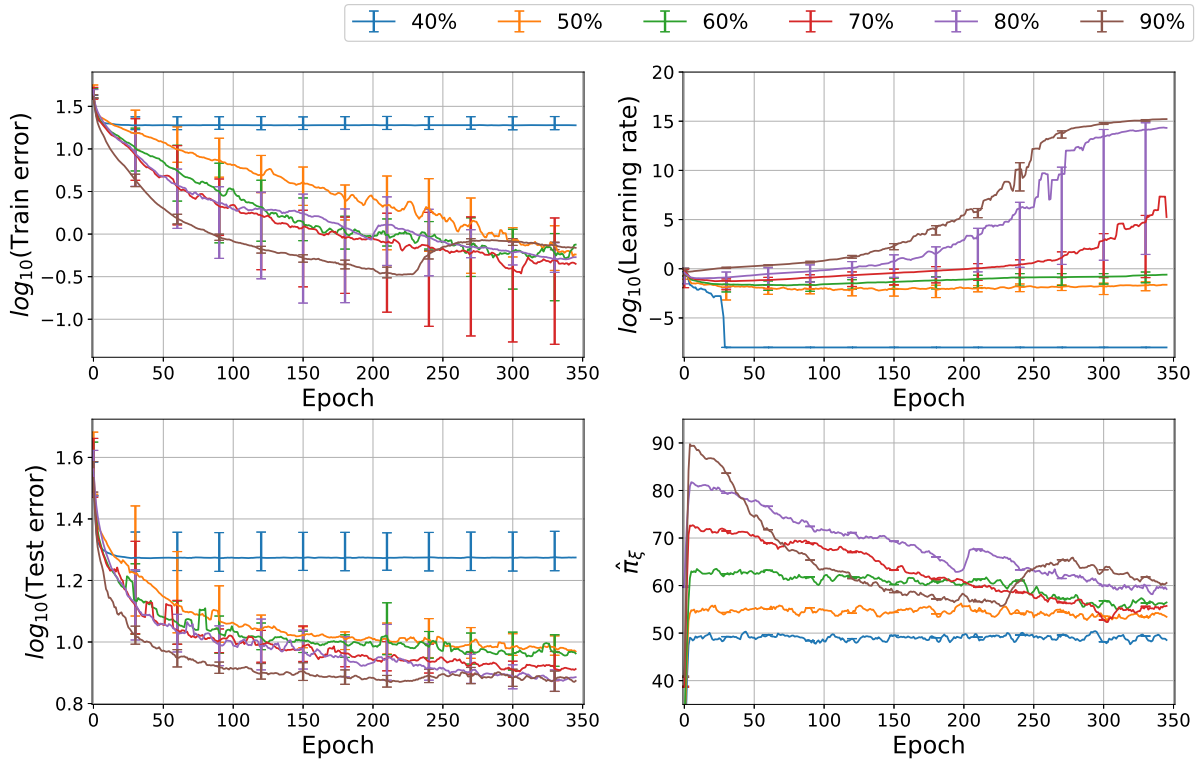


Figure 4.2: Various target probability hyperparameter, π_ξ, values tested for the GOCLS algorithm on the ResNet-18 architecture with CIFAR-10 using the SGD optimizer. Training error (top left), test error (bottom left), learning rates (top right) are shown on the $\log_{10}$ scale and the sample probability of positive signs, $\hat{\pi}_\xi$, (bottom right) against the number of epochs.

Choosing π_ξ for RMSProp

Figure 4.3 shows the training error, test error, learning rates on the $\log_{10}$ scale, and the sample probability of positive signs for the RMSProp optimizer. Similar to the behaviors observed in Figure 4.2, both training and test errors decline as the hyperparameter increases, resulting in larger learning rates. The higher the hyperparameter values become, the higher the growth in learning rates to maintain the sample probability, $\hat{\pi}_\xi$, as high as the target probability, π_ξ. We decided to choose the target probability, π_ξ, of 90% for RMSProp in the further experiments as it shows both the lowest training and test errors.

Choosing π_ξ for Adam

Figure 4.4 shows the training error, test error, learning rates on the $\log_{10}$ scale, and the sample probability, $\hat{\pi}_\xi$, of positive signs for the Adam optimizer. Both training and test errors decrease as the target probability, π_ξ, increases, however, with more than a probability of 70%, no significant difference in the test, training errors, and learning rates. Interestingly, the learning rate which converged towards when the target probability, π_ξ, is high is approximately the recommended learning rate for Adam. We decided to choose the target probability, π_ξ, of 70% for Adam in the experiments that follow, as it shows the lowest training and test errors.

Window size, ω

We employ GOCLS with a window size hyperparameter, ω, of 100. This section aims to briefly show the implication of the windows size hyperparameter to the training and test performances only using one of the optimizers we tested, SGD, with the target probability, $\hat{\pi}_\xi$, of 70%.

Figure 4.5 shows that the training, test errors, and learning rates on the $\log_{10}$ scale and the sample probabilities for SGD with different window sizes, including 50, 100, and 200. The

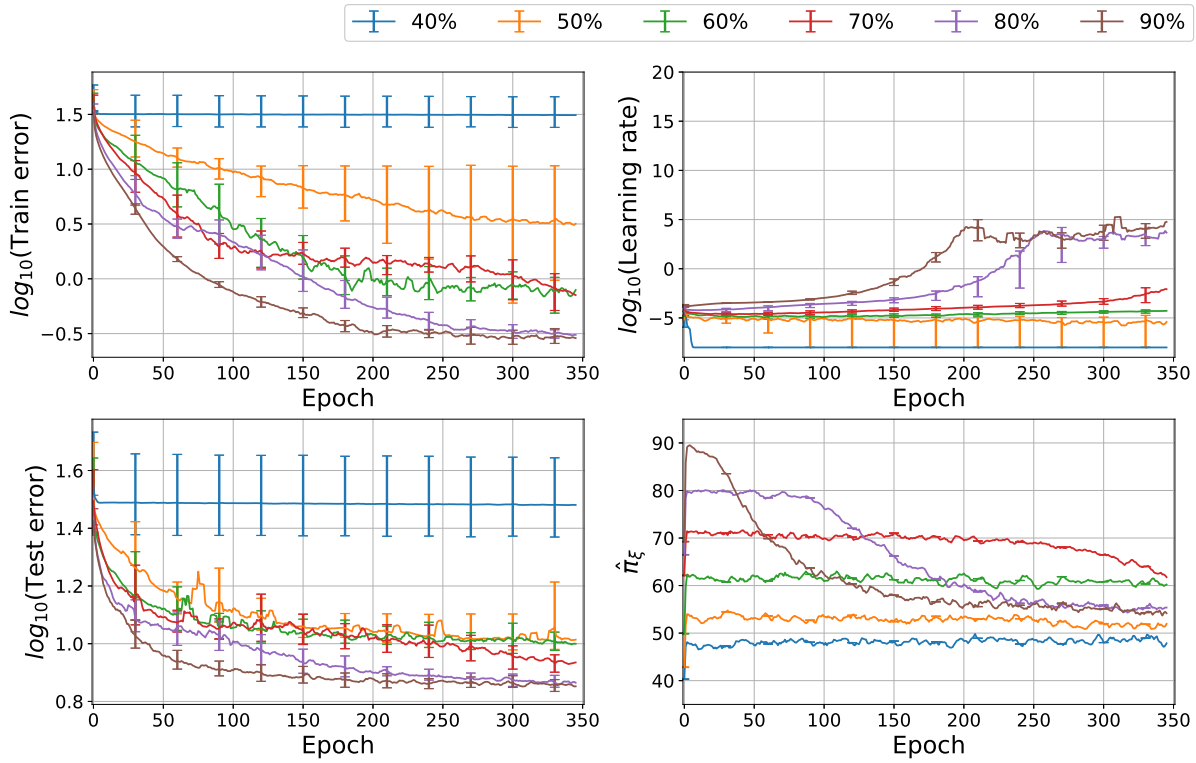


Figure 4.3: Various target probability hyperparameter, π_ξ, values tested for the GOCLS algorithm on the ResNet-18 architecture with CIFAR-10 using the RMSProp optimizer. Training error (top left), test error (bottom left), learning rates (top right) are shown on the $\log_{10}$ scale and the sample probability of positive signs, π_ξ, (bottom right) against the number of epochs.

training error of the smallest ω of 50 reduces at the quickest rate, and its learning rate is quick to increase. However, due to the instability, its training error abruptly increases close to the end of the training. On the other hand, the learning rate of $\omega = 200$ is hardly increasing, and as a result, the training error reduces at the slowest rate. The test errors of all three window size values are performing similarly. Hence, this motivates the choice of our $\omega = 100$, which is the intermediate value of the two extremes.

Conclusion of numerical study 1

The target probability value, π_ξ, is an interpretable hyperparameter across optimizers. As we increase the target probability value, we expect the learning rates to grow since it tries to overshoot, and subsequently, the optimizers outperform with a quicker convergence rate. When the target probability, π_ξ, is a high value, the sample probability, $\hat{\pi}_\xi$, tends to keep the learning rate large only initially, as it requires exponentially growing learning rates. The chosen target probabilities, π_ξ, for numerical experiments that follow are 70%, 90%, and 70% for SGD, RMSProp, and Adam, respectively.

4.6.2 Numerical study 2: Comparison between various learning rate strategies

This section compares the proposed Bayesian classification-based line search performance, GOCLS, against six other learning rate strategies discussed in Section 4.5.2, based on the relative robustness measure. We use the target probability hyperparameter, π_ξ, that produced the best results from the previous numerical study for different optimizers.

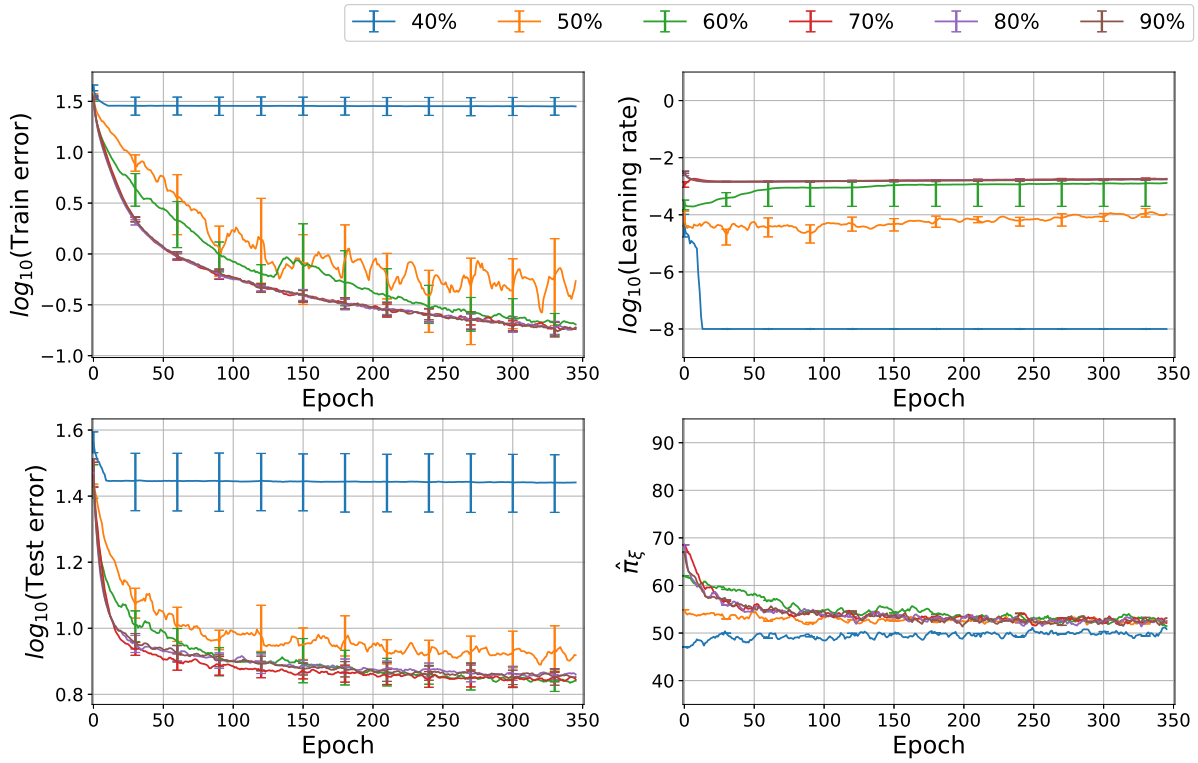


Figure 4.4: Various target probability hyperparameter, π_ξ, values tested for the GOCLS algorithm on the ResNet-18 architecture with CIFAR-10 using the ADAM optimizer. Training error (top left), test error (bottom left), learning rates (top right) are shown on the $\log_{10}$ scale and the sample probability of positive signs, $\hat{\pi}_\xi$, (bottom right) against the number of epochs.

ResNet-18

Figure 4.6(a) shows that both GOS and GOCLS have the highest training errors compared to the other strategies. The cosine annealing with $T_{max} = 350$ gets the lowest training error. However, both GOS and GOCLS certainly do not have the highest testing errors. GOS and GOALS-4, which have the highest overall learning rates, show the lowest test errors. We learn that decreasing the learning rate for convergence is the only way to obtain a good generalization result. The average number of gradient evaluations per iteration of GOS decreases since it tends to find convex quadratic approximation less frequently as the training continues. Conversely, that of GOALS-4 increases to satisfy the curvature condition [Chae et al., 2021]. The rest of the strategies only require a single number of gradient evaluations per iteration, which means more iterations are allowed per epoch with the same number of mini-batches.

Figure 4.6(b) indicates that overshooting with the RMSPROP optimizer outperforms the fixed learning rates and GOS. The cosine annealing with $T_{max} = 350$ obtains the lowest train. However, all the methods show similar test errors. Note that the learning rate of GOCLS increases exponentially, while that of the cosine annealing decreases exponentially.

Figure 4.6(c) shows that the fixed learning rate, GOS, and GOCLS produces almost identical training and test error as their learning rates are very similar to the recommended learning rate. The two cosine annealing and step decay show better results in ADAM's test and training errors in ResNet-18.

Table 4.2 shows the top average training and test accuracies and the differences, $\psi_{y,h,o}$, in the accuracies of various strategies, y, for each optimizer, o, on the problem, h, ResNet-18. The ratios of the test to training accuracies are given in the last column. The top training accuracies are shown by the step decay and cosine annealing strategies. However, all strategies have similar training performances, that the most considerable difference, $\psi_{y,h,o}$, observed in training accuracy is 0.36 for the fixed learning rate with the RMSPROP optimizer. The best

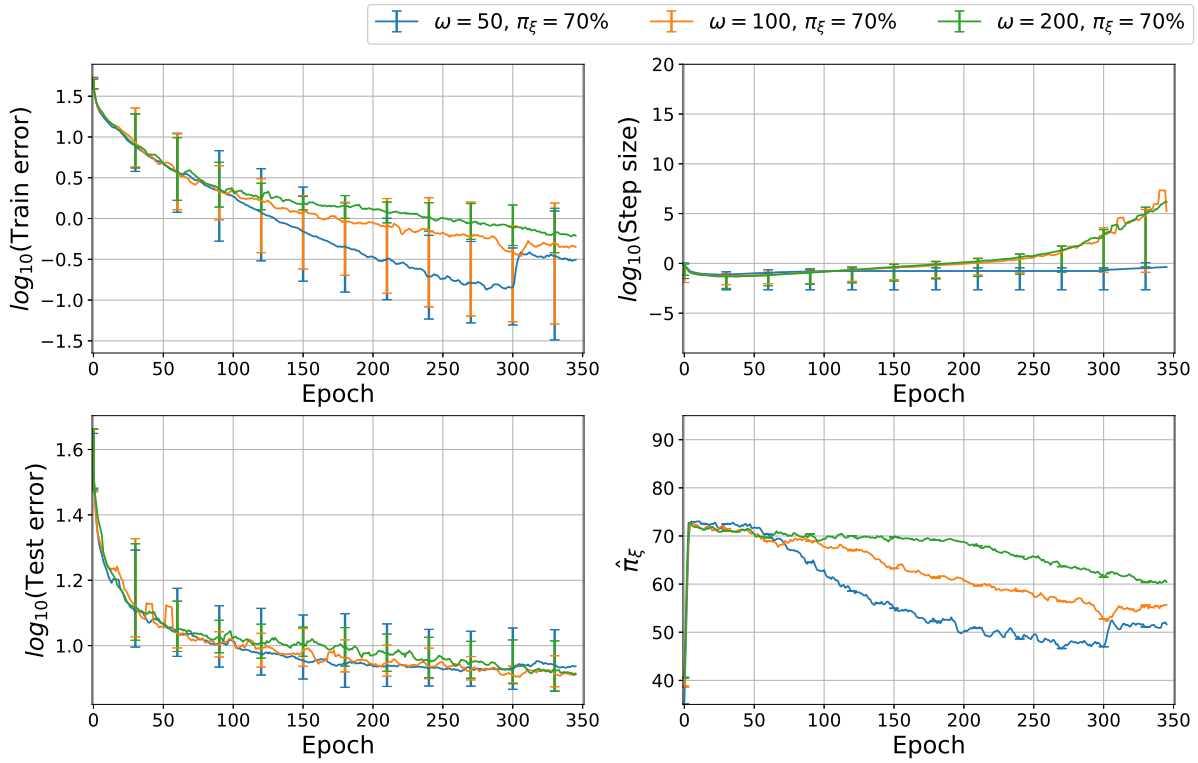


Figure 4.5: Various window size hyperparameter, ω, values tested for the GOCLS algorithm on the ResNet-18 architecture with CIFAR-10 using the SGD optimizer. Training error (top left), test error (bottom left), learning rates (top right) are shown on the $\log_{10}$ scale and the sample probability of positive signs, $\hat{\pi}_\xi$, (bottom right) against the number of epochs.

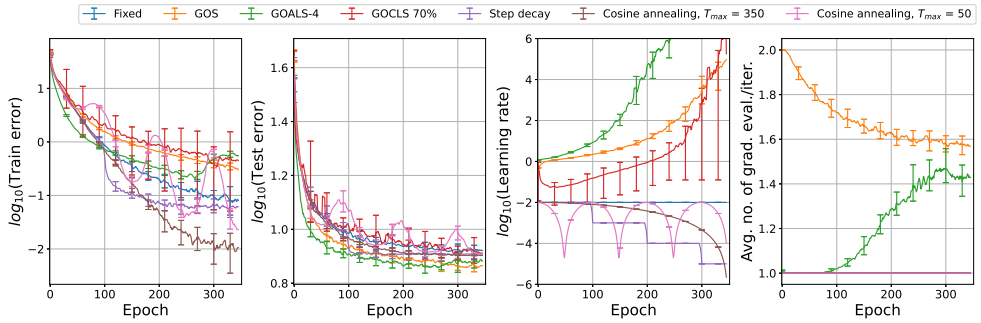
performing in terms of test accuracies for SGD, RMSProp, and Adam are GOALS-4, cosine annealing with $T_{max} = 350$, and step decay, respectively. The relative robustness measures, $R_{y,h}$, show that cosine annealing with $T_{max} = 350$ ranks first in both training and test robustness measures over the optimizers. On the other hand, GOCLS ranked last and fifth among the seven strategies. In terms of generalizability, GOCLS ranks second for both SGD and RMSProp and fifth for Adam.

EfficientNet-B0

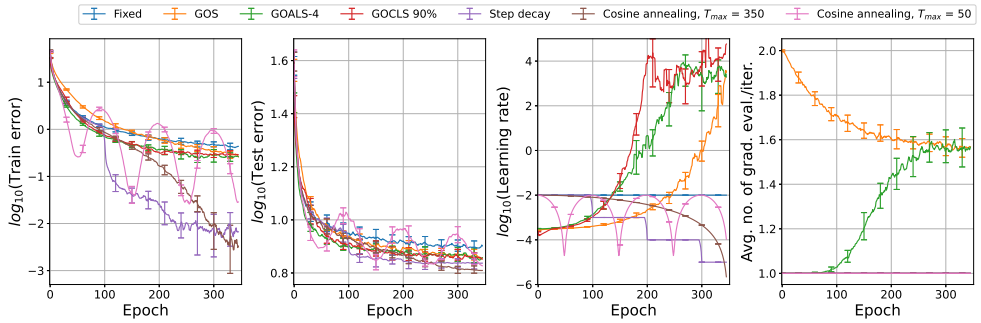
Figure 4.7 shows the performances of the various learning rate strategies on the EfficientNet-B0 architecture. Figure 4.7(a) shows that SGD with GOALS-4 has the lowest training error, followed by GOCLS. Their test errors are also significantly lower than the rest—the average learning rate of GOS about ten times larger than GOCLS.

Figure 4.7(b) shows that RMSProp results with GOCLS at the target probability of 90% and well-tuned GOALS-4 have an aggressive training aspect initially, and later, cosine annealing with $T = 350$ overtook it. The test error of GOCLS shows an average performance relative to the other methods. Figure 4.7(c) shows that although Adam with GOCLS again has the highest training error, its test error is on the average of all methods.

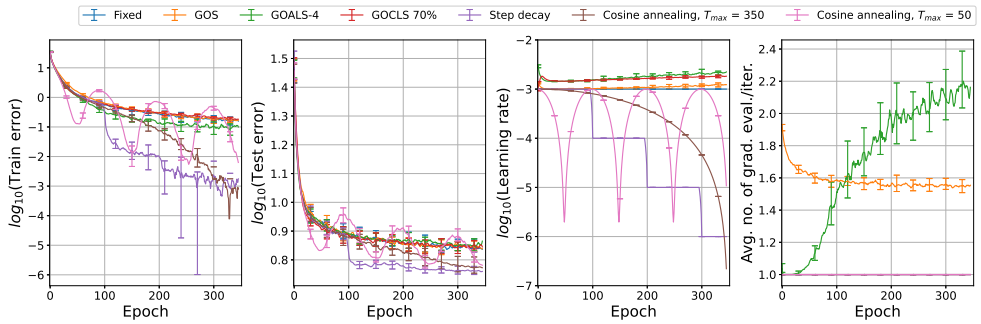
Table 4.3 shows the top average training and test accuracies and the differences, $\psi_{y,h,o}$, in the accuracies of various strategies, y, for each optimizer, o, on the problem, h, EfficientNet-B0. The ratios of the test to training accuracies are given in the last column. The highest training and test accuracies are given by GOCLS and GOS, respectively. The highest training and test accuracies for both RMSProp and Adam are given by cosine annealing with $T_{max} = 350$. However, the cosine annealing had large difference, $\psi_{y,h,o}$, values for training and test accuracies for SGD. Hence, although the cosine annealing gives the highest relative training robustness measures over the optimizers, the highest test accuracies are obtained by GOS, followed by GOALS. In



(a) SGD



(b) RMSProp



(c) Adam

Figure 4.6: ResNet-18: performance comparison for various learning rate strategies including the fixed learning rate, GOS, GOALS-4, GOCLS, step decay, cosine annealing with $T_{max} = 350$ and 50 using the CIFAR-10 dataset for (a) SGD, (b) RMSProp and (c) Adam. We present training error, testing error, learning rate on the $\log_{10}$ scale, and the average number of gradient evaluations per iteration for each optimizer.

terms of generalizability, GOCLS ranks third, seventh and first for SGD, RMSProp, and Adam, respectively.

Table 4.4 shows the overall training and test relative robustness measures, R_y, and the average ratios of test to training accuracies for the strategies. Cosine annealing with $T_{max} = 350$ and GOALS-4 rank first in the overall training and test relative robustness measures, R_y, respectively.

GOCLS ranks fourth in both training and test robustness measures. The two cosine annealing strategies and GOALS-4 lead GOCLS in the training robustness. GOS, cosine annealing with $T_{max} = 350$, and GOALS-4 lead GOCLS in the test results. The average ratios show that the generalizability of GOCLS ranks fourth, led by GOS, step decay, and GOALS-4. The least generalizable strategy or overfitting strategy is the fixed learning rate. Although the performance of GOCLS may not be the best in any of the categories, GOCLS certainly shows competitive overall robustness measures, R_y.

Moreover, the T_{max} value of cosine annealing $T_{max} = 350$, which showed the best training performance, was an unusual choice for the hyperparameter, which happens to perform well,

Optimizers	Strategies	Train acc. [%]	Diff., $\psi_{y,h,o}$	Test acc. [%]	Diff., $\psi_{y,h,o}$	Te./Tr.
SGD	Fixed	99.94	0.06	91.88	1.14	0.919
	GOS	99.72	0.28	92.93	0.09	0.932
	GOALS	99.8	0.2	**93.02**	0	0.932
	GOCLS	99.7	0.3	92.36	0.66	0.926
	Step	99.95	0.05	92.0	1.02	0.92
	Cosine, $T_{max} = 350$	**100.0**	0	92.08	0.94	0.921
	Cosine, $T_{max} = 50$	99.98	0.02	91.97	1.05	0.92
RMSprop	Fixed	99.64	0.36	92.37	1.25	0.927
	GOS	99.78	0.22	93.02	0.6	0.932
	GOALS	99.79	0.21	93.01	0.61	0.929
	GOCLS	99.75	0.25	93.03	0.59	0.933
	Step	**100.0**	0	93.22	0.4	0.932
	Cosine, $T_{max} = 350$	**100.0**	0	**93.62**	0	0.936
	Cosine, $T_{max} = 50$	99.98	0.02	93.32	0.3	0.933
Adam	Fixed	99.87	0.13	93.3	0.99	0.934
	GOS	99.86	0.14	93.23	1.06	0.934
	GOALS	99.94	0.06	93.16	1.13	0.932
	GOCLS	99.84	0.16	93.38	0.91	0.935
	Step	**100.0**	0	**94.29**	0	0.943
	Cosine, $T_{max} = 350$	**100.0**	0	94.12	0.17	0.941
	Cosine, $T_{max} = 50$	**100.0**	0	93.98	0.31	0.94
$R_{y,h}$	Fixed	-	0.55	-	3.29	-
	GOS	-	0.64	-	1.66	-
	GOALS	-	0.47	-	1.74	-
	GOCLS	-	0.71	-	2.07	-
	Step	-	0.05	-	1.33	-
	Cosine, $T_{max} = 350$	-	**0**	-	**1.02**	-
	Cosine, $T_{max} = 50$	-	0.04	-	1.57	-

Table 4.2: ResNet-18. The maximum mean values for both training and test accuracies over the five runs are listed for SGD, RMSProp and Adam, for the fixed learning rates, GOS, GOALS-4, step decay, cosine annealing with $T_{max} = 50$ and 350 on ResNet-18. $R_{y,h}$ presents the sum of the differences, $\psi_{y,h,o}$, between the accuracies and the best one for each optimizer. The maximum accuracies and the best robustness measured are shown in bold. The ratios of training to test accuracies are listed in the last column.

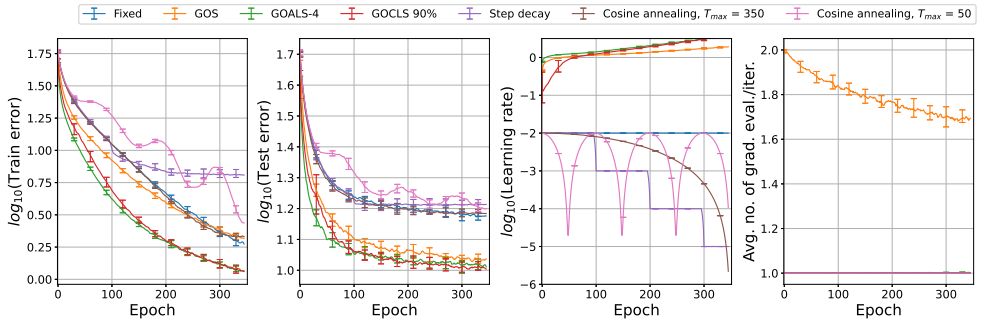
and GOS, which showed the best test performance, does not guarantee its convergence.

4.7 Conclusion

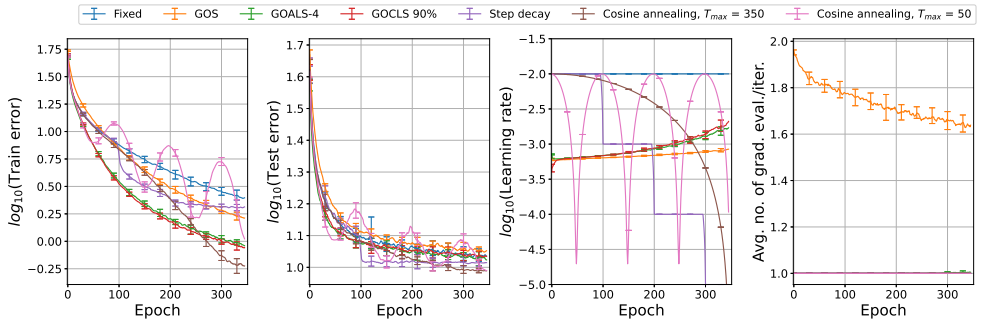
Learning rate is a crucial hyperparameter to be chosen carefully for successful training in neural network problems. Line search algorithms select learning rates for a descent direction provided by an optimizer, often requiring more than one function evaluation per iteration to satisfy specific optimality criteria. Hence, we proposed a computationally cost-efficient Bayesian classification-based line search method that uses the historical gradient data to choose learning rates by predicting the location of sign changes in directional derivatives. This approach only requires one gradient computation per line search iteration. Hence, there is no other gradient that we need to compute but for the optimizers.

GOCLS constructs bound approximations to resolve learning rates that provide the probability of observing positive directional derivative signs at a target probability. This target probability is a hyperparameter that can adjust the degrees of overshooting or undershooting. We conduct a hyperparameter study to decide which target probability would be appropriate for the optimizers, including SGD, RMSProp, and Adam. The hyperparameter study shows that the tested optimizers prefer overshooting to undershooting with high target probabilities: 70%, 90%, 70% of observing positive signs for SGD, RMSProp, and Adam, respectively.

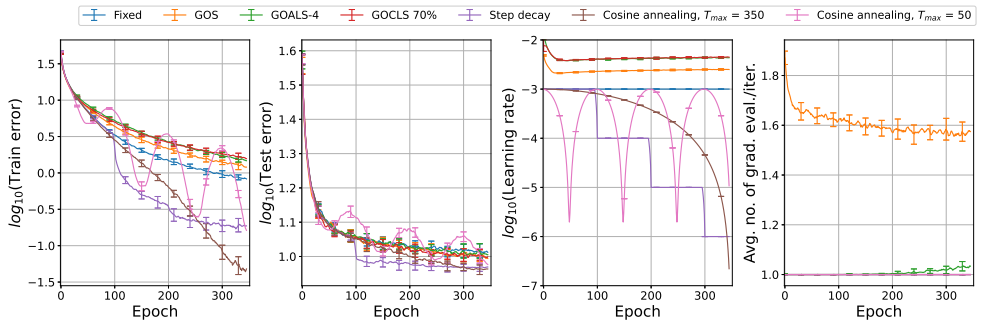
Using those target probability values, we conduct comparison tests for GOCLS with the various optimizers against six other learning rate strategies on ResNet-18 and EfficientNet-B0. We quantify their performances using the relative robustness measures that we propose. While the traditional metrics quantify only the best performances of strategies, the robustness measures consider all performances, including the best and poorest performances. As a result,



(a) SGD



(b) RMSProp



(c) Adam

Figure 4.7: EfficientNet-B0: performance comparison for various learning rate strategies including the fixed learning rate, GOS, GOALS-4, GOCLS, step decay, cosine annealing with $T_{max} = 350$ and 50 using the CIFAR-10 dataset for (a) SGD, (b) RMSProp and (c) Adam. We present training error, testing error, learning rate on the $\log_{10}$ scale, and the average number of gradient evaluation per iteration for each optimizer.

the measures favor well-performing strategies across different problems and optimizers rather than the strategies specialized at specific problems.

The results showed that GOCLS ranks fourth for both training and test relative robustness measures among seven strategies led by cosine annealing, GOS, and GOALS-4. However, the cosine annealing required a carefully selected and unusual value for the hyperparameter. GOS and GOALS-4 demonstrated the competitiveness of quadratic approximations. The results showed that the approach warrants further investigation in training DNN.

Optimizers	Strategies	Train acc. [%]	Diff., $\psi_{y,h,o}$	Test acc. [%]	Diff., $\psi_{y,h,o}$	Te./Tr.
	Fixed	98.18	0.7	85.37	4.48	0.87
	GOS	97.94	0.94	89.44	0.41	0.913
	GOALS	**98.88**	0	**89.85**	0	0.909
SGD	GOCLS	98.82	0.06	87.07	2.78	0.881
	Step	93.68	5.2	83.83	6.02	0.895
	Cosine, $T_{max} = 350$	97.93	0.95	84.86	4.99	0.867
	Cosine, $T_{max} = 50$	97.33	1.55	84.32	5.53	0.866
	Fixed	97.6	1.84	89.44	0.96	0.916
	GOS	98.43	1.01	89.0	1.4	0.904
	GOALS	99.13	0.31	89.59	0.81	0.904
RMSprop	GOCLS	99.16	0.28	89.38	1.02	0.901
	Step	98.04	1.4	89.83	0.57	0.916
	Cosine, $T_{max} = 350$	**99.44**	0	**90.4**	0	0.909
	Cosine, $T_{max} = 50$	98.99	0.45	90.3	0.1	0.912
	Fixed	99.21	0.75	89.96	0.96	0.907
	GOS	98.85	1.11	90.22	0.7	0.913
	GOALS	98.57	1.39	90.18	0.74	0.915
Adam	GOCLS	98.48	1.48	90.25	0.67	0.916
	Step	99.85	0.11	90.79	0.13	0.909
	Cosine, $T_{max} = 350$	**99.96**	0	**90.92**	0	0.91
	Cosine, $T_{max} = 50$	99.85	0.11	90.56	0.36	0.907
	Fixed	-	3.29	-	6.4	-
	GOS	-	3.06	-	2.51	-
	GOALS	-	1.7	-	**1.55**	-
$R_{y,h}$	GOCLS	-	1.82	-	4.47	-
	Step	-	6.71	-	6.72	-
	Cosine, $T_{max} = 350$	-	**0.95**	-	4.99	-
	Cosine, $T_{max} = 50$	-	2.11	-	5.99	-

Table 4.3: EfficientNet-B0. The maximum mean values for both training and test accuracies over the five runs are listed for SGD, RMSProp and Adam, for the fixed learning rates, GOS, GOALS-4, step decay, cosine annealing with $T_{max} = 50$ and 350 on EfficientNet-B0. $R_{y,h}$, presents the sum of the differences, $\psi_{y,h,o}$, between the accuracies and the best one for each optimizer. The maximum accuracies and the best robustness measured are shown in bold. The ratios of training to test accuracies are listed in the last column.

Strategies	Train overall, R_y	Train rank	Test overall, R_y	Test rank	Avg. Te./Tr.
Fixed	3.84	6	9.78	7	0.912
GOS	3.7	5	4.26	2	0.921
GOALS-4	2.17	3	**3.29**	1	0.921
GOCLS	2.53	4	6.63	4	0.915
Step	6.76	7	8.14	6	0.919
Cosine, $T_{max} = 350$	**0.95**	1	6.1	3	0.914
Cosine, $T_{max} = 50$	2.15	2	7.65	5	0.913

Table 4.4: The overall relative robustness measure, R_y, is computed by summing for ResNet-18 and EfficientNet-B0 in Tables 4.2 and 4.3. The best robustness measures, R_y, for training and test accuracies are shown in bold. The average ratios of training to test accuracies over the two problems are listed in the last column.

Chapter 5

Sub-dimensional Surrogates to Solve High Dimensional Optimization Problems in Machine Learning

5.1 Chapter overview

Approximation or surrogate models are widely used in optimization in either 1-D or full-dimensional models. While a 1-D approximation is a computationally inexpensive approach, it might not allow enough complexity to express the original problem for quick convergence. On the other hand, while a full-dimensional surrogate model allows for fast convergence and a more definite expression of the actual problem, this approach might require multiple computationally costly function evaluations for a high-dimensional problem. For optimizing high-dimensional, nonlinear, black-box problems, such as neural network problems, there needs to be a trade-off between increasing the dimensionality for complexity and reducing the computational cost in surrogate-based optimization. We try to achieve this balance by introducing sub-dimensional surrogates, in short, subsurrogates. The dimensionality of the subsurrogate is between one and the full dimension of the original problem, often much less than the full dimension. Since we search for the potential in the subsurrogate-based optimization approach, we intentionally remove any advanced features from the approach. This means that we employ full-batch sampling instead of mini-batch sampling, function-value-only surrogates instead of involving gradient information. Still, we add flexibility to the model by utilizing the cubic radial basis functions. We tested this crude subsurrogate approach on the Ackley and sum of squares functions for various data points, dimensions, and search directions. Lastly, we examine the approach on a simple single-layer neural network problem with various choices in dimensions. The results showed that the approach has merit in training DNN and deserves future research.

5.2 Introduction

High dimensional optimization problems, $f : \mathbb{R}^w \to \mathbb{R}$, $w \gg 100$, are either solved by successively solving a series of one-dimensional problems, which includes classical gradient-based optimizers, or by successively searching in the full w-dimensional space as is usually done by evolutionary approaches [Yang et al., 2008a]. As a consequence surrogate-assisted optimization [Queipo et al., 2005, Shyy et al., 2011, Shan and Wang, 2010, Kubicek et al., 2015] for high dimensional optimization problems has received limited attention, and is considered to be computationally intractable [Shan and Wang, 2010]. Recent attempts to address some of these challenges include constructing multiple multivariate linear interpolation models over different sub-spaces of a 16 dimensional design domain [Kang et al., 2017], to predict low cycle fatigue life. Constrained Optimization By Radial basis function Approximation (COBRA) has been used to solve a 124 dimensional vehicle dynamics problem subjected to 68 black-box inequality constraints [Regis, 2014]. The construction of surrogate models can be enhanced with gradient information, e.g.

weighted gradient-enhanced kriging (WGEK) [Han et al., 2013] has been used to combine a number of sub-domain models to solve a 108 design variable problem for the inverse design of a transonic wing [Han et al., 2017]. In turn, DYnamic COordinate search Response Surface models (DYCORS) construct surrogate models for the full dimension but only selects a subset of variables to be perturbed for additional DOE points to extend surrogate-based optimization to 200 dimensional problems [Regis and Shoemaker, 2013].

As an alternative way to solve optimization problems in sub-spaces, the successive one-dimensional updates of coordinate descent (CD), random coordinate descent (RCD) [Nesterov, 2012] and cyclic coordinate descent (CCD) [Canutescu and Dunbrack Jr, 2003, Sauer and Bouman, 1993, Bertsekas, 1997, Wright, 2015] has been extended to block coordinate descent (BCD) by selecting a subset or block of variables [Luo and Tseng, 2002, Richtárik and Takáč, 2014] to update. That is a high dimensional problem solved by successively solving sub-dimensional problems. In evolutionary optimization, this concept was introduced as cooperative searches by Potter and Jong (1994) [Potter and De Jong, 1994], and dynamically dimensioned searches (DDS) by Tolson and Shoemaker (2007) [Tolson and Shoemaker, 2007]. Renewed interest into CD and BCD has been sparked by success in various applications in statistical learning [Friedman et al., 2008, 2010] and machine learning [Chang et al., 2008].

Research into CD and BCD has been largely limited to direct optimization of objective functions with limited work on surrogate-based optimization, which are limited to the work by Werth et al. [2017] that divides high-dimensional optimization problems into low-dimensional sub-problems using the monotony and non-linearity tests to solve 1000 dimensional problems. A sub-dimensional surrogate model can be optimized using appropriate optimizers that may include gradient-based, gradient-free or evolutionary optimizers [Regis and Shoemaker, 2013, Sun et al., 2017]. A number of research questions in sub-dimensional searches remains, which includes:

1. Selection of sub-dimensional spaces: cyclically, randomly, elastic nets [Zou and Hastie, 2005], DM-HDMR [Mahdavi et al., 2014], monotony [Werth et al., 2017], non-linearity [Werth et al., 2017], random decomposition method (DECC-G) [Yang et al., 2008a] and the self-adaptive decomposition method [Yang et al., 2008b]; 2. Number of variables to define the sub-space; 3. Which sub-space to select; 4. Appropriate test problems. In this study, we investigate in particular the utility of the gradient vector to select sub-dimensional spaces for a vanilla algorithm that incorporates no heuristics. The availability of analytical gradients is becoming standard with automatic differentiation numerical libraries such as Pytorch and Tensorflow [Paszke et al., 2017, Shukla and Fricklas, 2018]. We demonstrate that the gradient vector allows for sensible selection of sub-space over which to construct surrogate models for 500 dimensional test problems that include Ackley and Sum of squares, in both their original description frames and rotated description frames. We include the performance of CD and steepest gradient descent (SGD) as baseline strategies. Various potential heuristics are proposed based on the lessons learned from the test problems. Lastly, the sub-dimensional surrogate strategy was implemented to train the Iris dataset with 11, 51, and 507 weight variables in order to compare the relationships between the training errors and test errors for a various number of dimensions for sub-dimensional surrogates.

5.3 Sub-dimensional Surrogates

Given a function $f : \mathbb{R}^w \to \mathbb{R}$, we may construct an $v \ll w$ dimensional surrogate function $\hat{f}_n : \mathbb{R}^v \to \mathbb{R}$ that approximates $f : \mathbb{R}^v \to \mathbb{R}$ with the remaining variables $\boldsymbol{x}_n^{w-v}$ kept fixed. Numerous approaches have been proposed for CD and BCD strategies to select the v variables, which could be used to construct an v-dimensional surrogate. These include cyclically, randomly, elastic nets [Zou and Hastie, 2005], DM-HDMR [Mahdavi et al., 2014], monotony [Werth et al., 2017], non-linearity [Werth et al., 2017], random decomposition method (DECC-G) [Yang et al., 2008a] and the self-adaptive decomposition method [Yang et al., 2008b]. In this study, we investigate the gradient vector to serve as a selection strategy, with gradients that are becoming more and

more accessible with the likes of automatic differentiation frameworks such as Tensorflow and PyTorch that are used extensively in statistical learning, machine learning, deep learning, data science and numerical modelling.

5.3.1 Sub-dimensional Greedy Surrogates

Given the current best solution $\boldsymbol{x}_n$, we compute the gradient $\boldsymbol{\nabla} f(\boldsymbol{x}_n)$ and select the v gradient components with the largest magnitudes to select the sub-space $\mathbb{R}^v$ in which we construct our v-dimensional surrogates, i.e. we construct greedy v-dimensional surrogates following the four steps:

1. Compute $\boldsymbol{\nabla} f(\boldsymbol{x}_n \in \mathbb{R}^w)$;
2. Select the $v \ll w$ variables, $\boldsymbol{v}$, associated with the v largest components of $\boldsymbol{\nabla} f(\boldsymbol{x}_n)$;
3. Sample the v-dimensional sub-space to obtain p new samples, while keeping the other variables $\boldsymbol{x}_n^{(w-v)}$ fixed;
4. Using the p sampled designs, construct the sub-dimensional surrogate $\hat{f}_n : \mathbb{R}^v \to \mathbb{R}$;
5. Find the minimizer $\boldsymbol{x}_n^{v*} \in \mathbb{R}^v$ by minimizing $\hat{f}_n$;
6. Update $\boldsymbol{x}_{n+1}^v = \boldsymbol{x}_n^{v*}$ to construct $\boldsymbol{x}_{n+1} \in \mathbb{R}^w$;
7. Repeat steps 1 - 6 until convergence or maximum iterations is reached.

As this is one of the first papers to investigate sub-dimensional surrogates to solve high dimensional problems, our aim is not to propose state of the art algorithms but rather to investigate the potential merits of sub-dimensional surrogates to solve high dimensional problems. In particular, we want to investigate the merits of using the gradient vector as a variable selection strategy. Hence, we only consider a base (or vanilla) algorithm without the incorporation of heuristics for this study. This is done to allow us to explore the merits of solving sub-dimensional surrogates, with a gradient-based selection criterion. Although we do not include heuristics, we do offer a number of avenues that could be pursued to include heuristics that offers a substantial improvement on the performance of the current study that could be explored in future studies.

Construction of Sub-dimensional Surrogates

At the n^{th} iterate, given that $\boldsymbol{x}_n^{w-v} \in \mathbb{R}^{w-v}$ remains fixed, we construct an v-dimensional surrogate $\hat{f}_n : \mathbb{R}^v \to \mathbb{R}$. It is evident that information density over which we construct a surrogate is critical i.e. larger domains require more sampling points for the same accuracy. Although this is important, we simplify our study by fixing the sampling of surrogate domain $\boldsymbol{D}_s$ to 120% of the original bound constrained domain $\boldsymbol{D} = \{x_i^{min} \leq D_i \in \mathbb{R} \leq x_i^{max}, i \in \boldsymbol{v}\}$. The bound constraints are however imposed to be the size of the original domain, this is to ensure we have a proper surrogate description at the bounds of the problem.

We, therefore, sample the sub-dimensional surrogate domain using Latin Hypercube Sampling (LHS) [McKay et al., 2000] using p points. For the surrogate construction, we transform $\boldsymbol{D} = \{x_i^{min} \leq D_i \in \mathbb{R} \leq x_i^{max}, i \in \boldsymbol{v}\}$ to $\boldsymbol{N} = \{0 \leq N_i \in \mathbb{R} \leq 1, i \in \boldsymbol{v}\}$. Hence, before evaluating the surrogate, we first transform $\boldsymbol{x} \in \boldsymbol{D}_s$ to $\boldsymbol{z} \in \boldsymbol{N}$.

In this study, we construct radial basis function (RBF) surrogate supplemented by a bias and linear polynomials [Powell, 1992, 2005].

$$\hat{f}(\boldsymbol{x}) = \begin{bmatrix} 1 & \boldsymbol{x} \end{bmatrix} \begin{bmatrix} \boldsymbol{c}_1 \\ \boldsymbol{c}_2 \\ \vdots \\ \boldsymbol{c}_p \end{bmatrix} + \Sigma_{i=1}^{w} \boldsymbol{\lambda}_i \boldsymbol{\Phi}_i(\boldsymbol{x}), \tag{5.1}$$

using the cubic radial basis function

$$\phi(r_i(\boldsymbol{x})) = r_i(\boldsymbol{x})^3, r_i = \|\boldsymbol{x} - \boldsymbol{x}_i\|_2, \tag{5.2}$$

where $r_i(\boldsymbol{x})$ is the Euclidean distance $\boldsymbol{x}$ to the center $\boldsymbol{x}_i$ of the $\boldsymbol{i}^{th}$ basis function. Given $p = v$ sampled LHS designs $\boldsymbol{X} \in \mathbb{R}^{v \times p}$, with their associated function evaluations, $\boldsymbol{f}$, the unknown RBF weights $\boldsymbol{\lambda}$ and polynomial coefficients $\boldsymbol{c}$ are then solved from the following square system of equations

$$\begin{bmatrix} \boldsymbol{\Phi} & \boldsymbol{P} \\ \boldsymbol{P} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda} \\ \boldsymbol{c} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{0} \end{bmatrix}, \tag{5.3}$$

to obtain an interpolation based RBF approximation. Here, $\boldsymbol{\Phi}^{p \times p}$ denotes the matrix with each i^{th} row $\phi_i(\boldsymbol{x}_j)$ for

$$\boldsymbol{x}_j \in \boldsymbol{X}_j, j = 1, ..., p. \tag{5.4}$$

The cubic basis functions together with low order polynomials have been demonstrated by Regis and Shoemaker [2013] and Ilievski et al. [2017] of which the linear polynomial tail ensures for unique solutions if and only if the rank of the matrix $\boldsymbol{P}$ is $1 + w$ to obtain an invertible coefficient matrix in (5.3) [Powell, 1992].

Sub-dimensional Surrogate Minimization

The sub-dimensional surrogates (or 'sub-surrogates' for compactness) are then minimized using the Nelder-Mead algorithm [Nelder and Mead, 1965] by using MATLAB [2015] built-in optimization algorithm `fminsearch` from 10 random initial starting points within the feasible domain. Although we are solving 500-dimensional problems in this study for which Nelder-Mead is intractable, our sub-dimensional surrogates are limited to 1, 2 and 4-dimensions. The optimization is conducted from 9 random starting points, with the tenth point as the current best point.

The sub-dimensional minimizer $\boldsymbol{z}_n^{v*} \in \boldsymbol{N}$ is then transformed to $x_n^{v*} \in \boldsymbol{D}$ and the solution for the next iterate constructed $\boldsymbol{x}_{n+1} \in \mathbb{R}^w$.

5.4 Numerical Study Outline

In this study, we first consider two well-known test problems, listed in Table 5.1, with $n = 500$ (MATLAB codes for the test problems supplied by Surjanovic and Bingham (2013) [36]). We also include the performance of steepest gradient descent (SGD) and coordinate descent (CD) algorithms. In addition, we include results for the problems formulated in their proposed reference frame, as well as in an arbitrary rotated reference frame. This is to investigate the performance when the formulated functions are separable as well as when there is a strong interaction between the variables. That is the function $f(\boldsymbol{x})$ expressed in the original coordinate system $\boldsymbol{x} \in \mathbb{R}^w$ is related to the arbitrary rotated function $\acute{f}(\acute{\boldsymbol{x}})$ as follows $\acute{\boldsymbol{x}} = \boldsymbol{Q}\boldsymbol{x}$, where $\boldsymbol{Q} \in Orth^+$ is an arbitrary proper orthogonal matrix, i.e. $det(\boldsymbol{Q}) = 1$ as shown in Figure 5.1. In this study, $\boldsymbol{Q}$ is constructed following a simple series expansion of an exponential map $\boldsymbol{Q} = \boldsymbol{I} + \boldsymbol{W} + (1/2)\boldsymbol{W}\boldsymbol{W}\boldsymbol{W} + (1/6)\boldsymbol{W}\boldsymbol{W}\boldsymbol{W} + ...$ as outlined by Moler and Van Loan (2003) [37], where $\boldsymbol{W} = \frac{\alpha\pi}{180}(\boldsymbol{A} - \boldsymbol{A}^T)$ and $\boldsymbol{A}$ is a random $w \times w$ matrix, with each entry a random number between -0.5 and 0.5.

Function	$\boldsymbol{x}_{min}$	$\boldsymbol{x}_{max}$	$f(\boldsymbol{x}^*)$
Ackley	-32.768	32.768	0
Sum of squares	-5.12	5.12	0

Table 5.1: Prescribed domain and the global minimum function values for the two test functions.

Although the problems under consideration are test problems, some of them are representative of actual objective functions often encountered in machine learning and deep learning. In

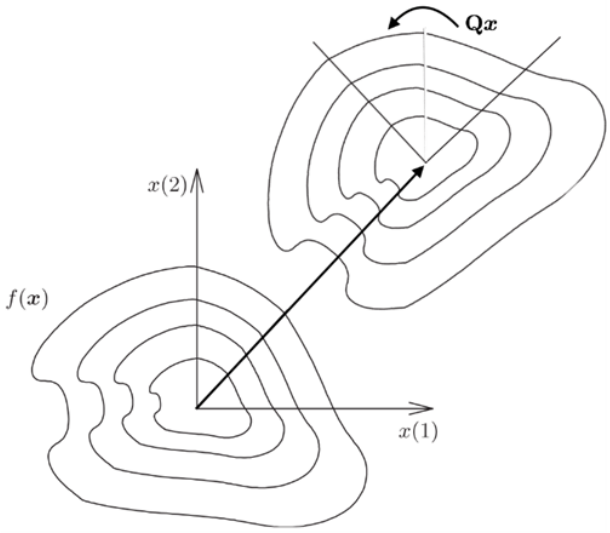


Figure 5.1: A function and the same function arbitrarily rotated to affect the variable interaction.

particular, the Ackley function is a good candidate for a typical state of the art deep network, namely ResNet-56 [Li et al., 2017], as depicted in Figure 5.2. It is well known that objective function surfaces in machine learning often have large domains with low gradients and curvatures, which surround higher curvature domains of the manifold [Goodfellow et al., 2014].

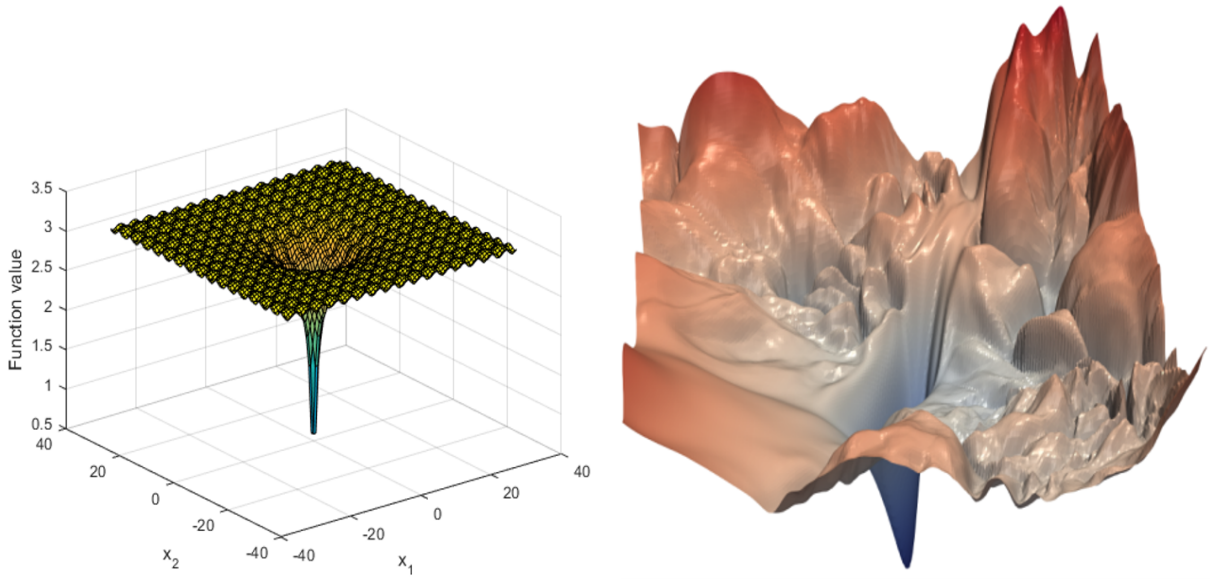
The remaining test problem, Sum of squares is representative of general ellipsoidal bowl-shaped optimization problem that is coercive. This test problem is representative of classical regularization or penalization that results in a coercive function given a sufficiently large regularization parameter. We conclude our numerical study with the neural network training a deep neural network on the Iris dataset to convey the basic findings of the potential impact of this approach on machine learning when considering not only a training set performance but also the test set performance for generalization of the trained result. Note that for the two test problems, the gradients were computed analytically to resemble gradients from automatic differentiation packages such as Tensorflow or Pytorch.

To allow for a systematic study, for both SGD and CD at every iteration, a fixed number of function evaluations were taken along the descent direction using Latin hypercube samplings (LHS) of the actual function. The update to the minimum function value was considered as the update step at every iteration. Hence, the number of function evaluations per iteration for SGD and CD matches the number of points sampled per iteration for a sub-dimensional surrogate approach to generate surrogate models. For CD, the highest magnitude partial derivative in the gradient is chosen as the coordinate to update. Lastly, the sampling domain is chosen to be 20% larger than the surrogate optimization domain along the descent direction.

Surrogate optimizers are denoted 1-D, 2-D, and 4-D, while SGD1 and SGD2 denote SGD sampling the univariate function along only the descent direction or sampling the univariate function along the entire sub-domain, i.e. along ascent and descent of the univariate function. Similarly, CD1 and CD2 respectively denote sampling along the descent direction or sampling over the entire domain of the univariate function. It is important to note, that differences between 1-D and CD2 are only due to the surrogate being minimized in 1-D using the same sampling points as CD2.

We limit the maximum number of function evaluations to 10^5, for the sub-dimension $v = 1, 2$ and 4, using $p = 8, 16, 32$ and 64 LHS sampling points per iteration. As we do not vary the size of the surrogate domain, the information density for each surrogate remains constant at every iteration.

The expected values for each test problem were computed over 30 runs. The initial guesses were taken randomly from the prescribed domains for each test problem as listed in Table 5.1.



(a) Ackley 2-D (b) ResNet-56

Figure 5.2: The Ackley function in 2-D (left) and the loss surfaces of ResNet-56 (right) [Li et al., 2017], both functions are depicted on a log-scale for the function values.

5.5 Results

The results for our two test functions, Ackley and Sum of squares are discussed in this section. Although it is expected for SGD to perform exactly the same for the unrotated and rotated problems, observed differences are due to the stochastic sampling along the search directions.

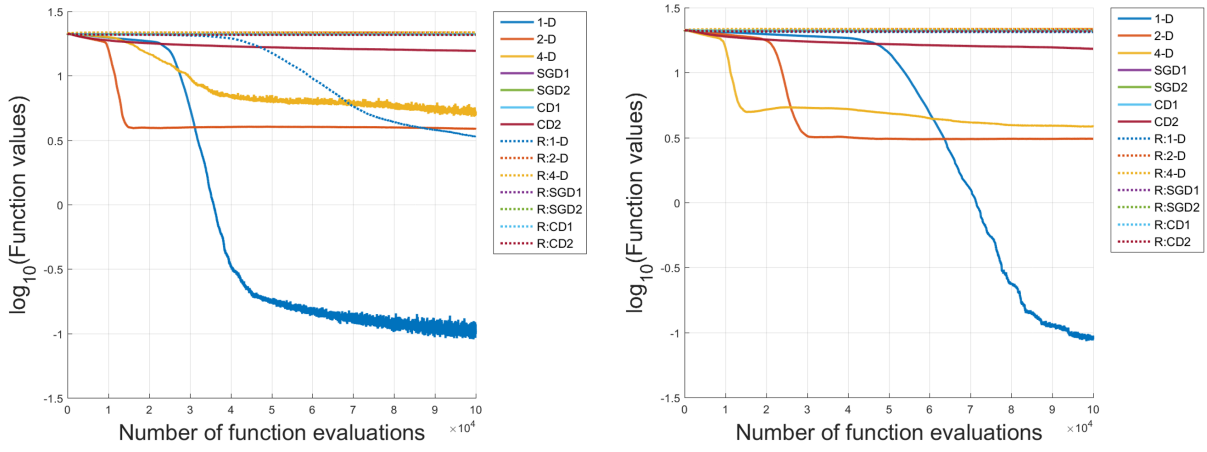
5.5.1 Ackley Results

In particular, the results of the various optimizers on the Ackley function are shown in Figure 5.3(a)-(d). It is clear that as the number of sampling points increases, the higher sub-dimensional surrogates are better defined and the respective variance of the 30 averaged solutions decrease, e.g. for the 4-dimensional sub-surrogate the averaged response using $p = 16$ sampling points is more smooth than the $p = 8$ LHS points.

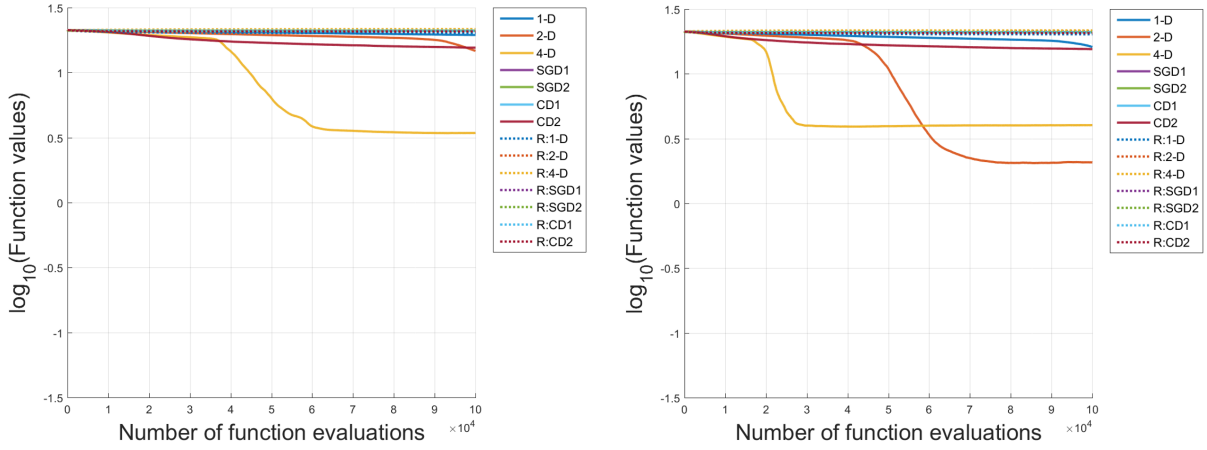
For $p = 8$ sampling points, the initial performance of the 2-dimensional sub-surrogate model performs the best. The performance stagnates in the limit as the accuracy of the surrogate is reached, since we do not reduce the volume of the domain over which the surrogate is constructed. The 1-D surrogates have a much higher sampling density and consequently result in a significantly more accurate surrogate that is limited to a single dimension.

For $p = 16$ sampling points, the initial performance of the 4-dimensional sub-surrogate model now performs the best, i.e. the higher sampling density results in a much more reliable surrogate to guide the optimizer. In fact, using $p = 16$ points for 4-dimensional sub-surrogate outperforms the 2-D sub-surrogate sampled using both $p = 8$ and $p = 16$ points. As expected, the performance stagnates in the limit as the accuracy of the surrogates reaches a limit, since we do not reduce the domain over which the surrogates are constructed. It is also evident that the surrogates become more accurate in the limit as more sample points are used, however in most cases to reach this accuracy a larger number of total function evaluations are required. The exceptions here are the 4-dimensional sub-surrogate that obtains better solutions using fewer total function evaluations when sampling $p = 16$ points as apposed to $p = 8$ samples, whereas $p = 32$ and $p = 64$ results in better solutions but requires significantly more total function evaluations to reach the same accuracy as $p = 8$ and $p = 16$ sampling points.

For $p = 32$ and 64 sampling points, it is evident that the required number of function evaluations to reach a specific accuracy is significantly more. The total improvement of the



(a) $p = 8$ (b) $p = 16$



(c) $p = 32$ (d) $p = 64$

Figure 5.3: Ackley function sampled with (a) $p = 8$, (b) $p = 16$, (c) $p = 32$, and (d) $p = 64$ data points at each iteration for unrotated (solid line) and rotated problem (dashed line) reference frames for the problem description.

solution in the limit due to a better-resolved surrogate is also marginal. In turn, reducing the problem dimension from 4-D to 2-D, and 2-D to 1-D significantly improves the accuracy of the sub-dimensional surrogates in the limit. This study clearly demonstrates that surrogates do not have to be either one-dimensional or full-dimensional, but sub-dimensional surrogates are powerful in that they allow for an alternative mechanism between exploration and exploitation, as well as adjusting sampling density given a fixed number of points. This adds an additional mechanism to control the sampling density, which traditionally consisted of:

1. Sampling using more points to better resolve a surrogate;
2. Sampling over a sub-domain (but in the same sub-space) is closely related to using more sampling points over sub-domain of the same volume, which this study complements with a significant third mechanism;
3. Reducing the sub-dimension of the surrogate, which significantly enhances the limit accuracy of the surrogate.

We consider a second test function to demonstrate that a significant amount of work is still required as the performance of Ackley, which seems to be well suited to a number of machine learning and deep learning objective functions, does not necessarily apply to all test problems. We, therefore, consider a second test problem namely the Sum of squares function that is a coercive function (as $\|\boldsymbol{x}\|_2 \to \infty$, $f(\boldsymbol{x}) \to \infty$), which is distinct from the noisy flat planes of the Ackley test function.

5.5.2 Sum of Squares Results

The results of the various optimizers on the Sum of squares function are shown in Figure 5.4(a)-(d). Two significantly different characteristic changes are evident. It is clear that coordinate descent performs much better on Sum of squares than Ackley.

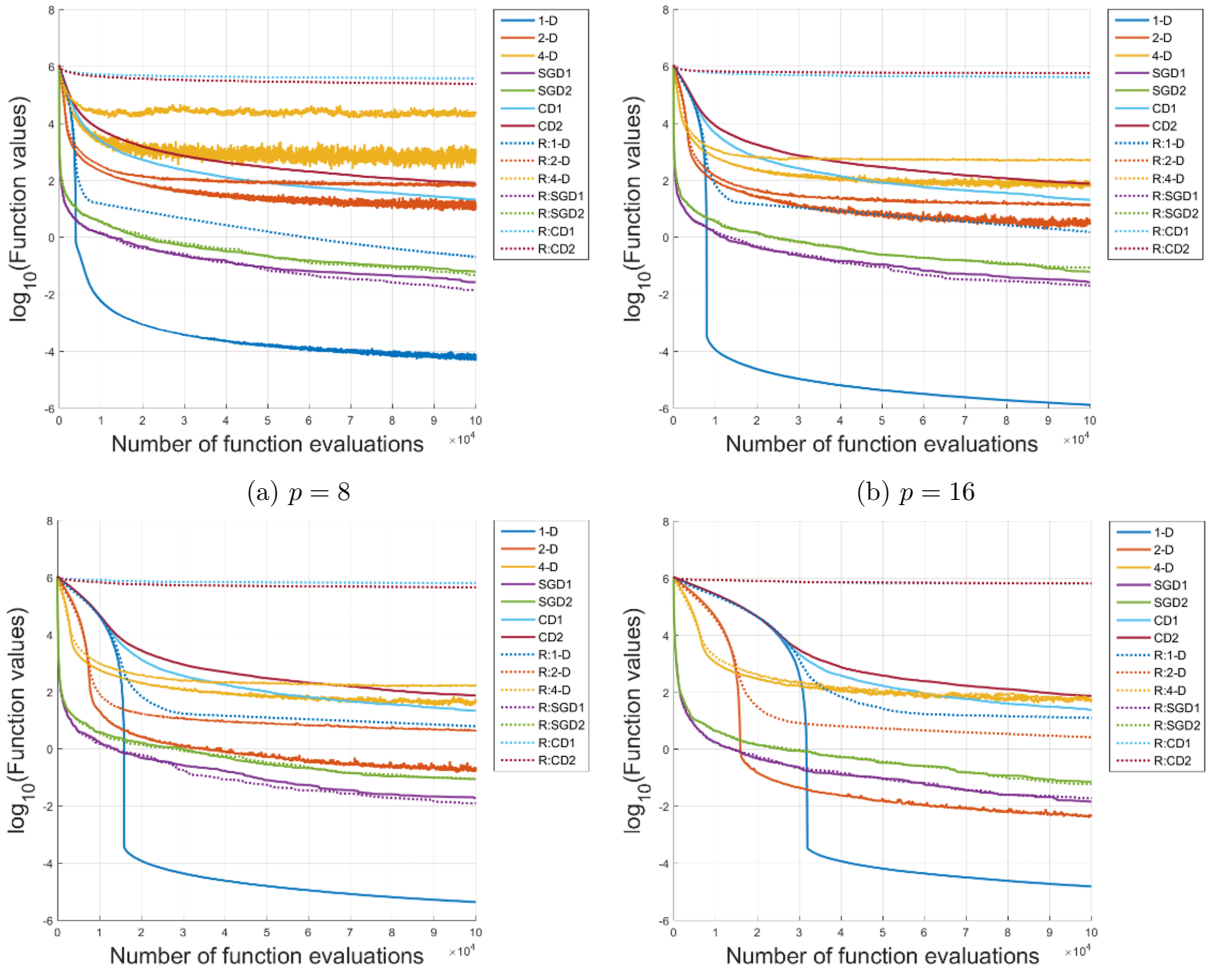
For $p = 8$ sampling points, the initial performance of SGD1 (descent direction domain sampling) and SGD2 (full domain sampling) and the 2-dimensional sub-surrogate models perform the best, which is followed by the 4-dimensional and 1-dimensional sub-surrogate models. The performance stagnates in the limit as the accuracy of the surrogate is reached.

For $p = 16$ sampling points, the initial performance of both CDs and 1-dimensional sub-surrogate models are similar, with a significant improvement in the lowest error of the 1-dimensional sub-surrogate model. The initial performance of the 4-D sub-surrogate model significantly outperforms the 2-D sub-surrogate but it stagnates quicker than 2-D sub-surrogate model.

For $p = 32$ and 64 sampling points, it is evident that the 2-D benefits from more sampling points to obtain lower minima due to a higher sampling density. Importantly, the 4-D sub-surrogate has the best initial performance of all the strategies, while the 1-D sub-surrogate sampled using 64-points has the best overall final performance. Note the larger the number of sampling points becomes the less iterations are achieved for a fixed number of function evaluations. Hence, the graphs look stretched as the rate of decrease in the model errors affecting the function value is slower than the rate of convergence of function value due to new models at each iteration.

Although the performance of the various algorithms on Sum of squares is significantly different to Ackley, the noteworthy observations based on Ackley are still evident on Sum of squares, i.e. higher dimensional surrogates have significantly better initial performance and reducing the surrogate sub-space significantly increases the sampling density.

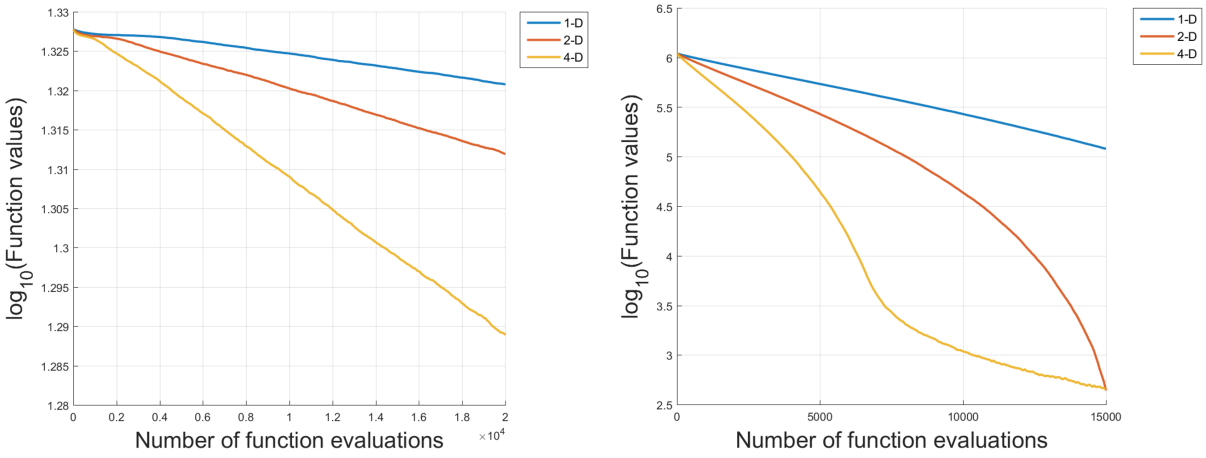
For unrotated (solid) and rotated (dashed) Sum of squares descriptions, it is clear that by increasing the interaction amongst the variables, the performance in general degrades. That said, it is important to note that the severity is more pronounced for the lower dimensional surrogates and alleviated for the higher dimensional surrogates. This is a critical observation:



(a) $p = 8$ (b) $p = 16$

(c) $p = 32$ (d) $p = 64$

Figure 5.4: Sum of squares function sampled with (a) $p = 8$, (b) $p = 16$, (c) $p = 32$, and (d) $p = 64$ data points at each iteration for unrotated (solid line) and rotated problem (dashed line) reference frames for the problem description.



(a) zoom-in view of Figure 5.3(d)

(b) zoom-in view of Figure 5.4(d)

Figure 5.5: Performance differences between surrogate sub-dimensions when sampling using $p = 64$ points on (a) Ackley, and (b) Sum of squares.

searching in sub-dimensions makes the optimization strategy less sensitive to stronger interaction or coupling amongst the variables.

Evidently, higher sub-dimensional surrogates allow for better exploration and initial improvement, while lower dimensional surrogates allow for surrogates that are better resolved. This opens up a new area of research to explore algorithms and heuristics that can be developed to explore and exploit the benefits and trade-offs of sub-dimensional surrogates.

5.5.3 Discussion on Higher Versus Lower Dimensional Searches on Initial Performance

We demonstrate that on all the test problems considered, the 4-D performs best, followed by 2-D, followed by 1-D sub-surrogates in Figure 5.5(a)-(b).

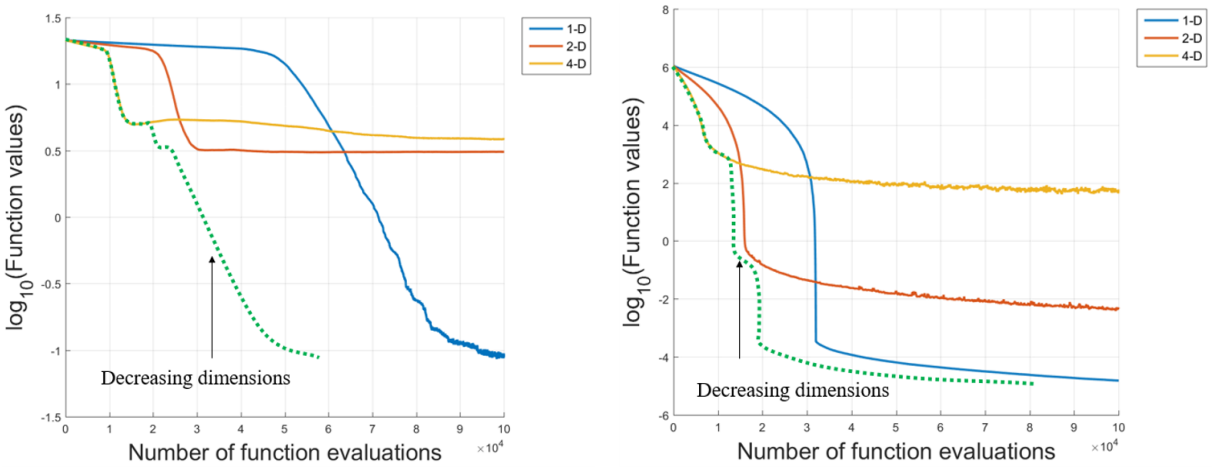
This clearly informs desirable characteristics for heuristics to be developed for sub-dimensional surrogates, i.e. initial searches should be high dimensional, while lower the dimensionality of the searches should be reduced as improvements are made. This is in addition to simple heuristics that would have significantly improved the performance of our sub-dimensional surrogates in this study but at the cost of obscuring insights due to the heuristic performance dominating the foundational algorithmic performance. For example, choosing the best solution not only from the best surrogate solutions but also from the best-sampled solutions. This would have completely hidden any indication of the quality of the surrogate surface. This just again emphasizes the importance of using a vanilla sub-dimensional surrogate strategy in initial investigations. We emphasize again this study does not propose a state of the art algorithm but clearly highlights insight and focus points for potential heuristics in this domain of sub-dimensional surrogates.

5.6 Lessons Learned and Sensible Heuristics

Performance of the proposed vanilla flavored sub-dimensional surrogates can be significantly improved by incorporating sensible heuristics. Based on the findings of our study, we outline a number of potentially sensible heuristics as well as what they should aim to achieve.

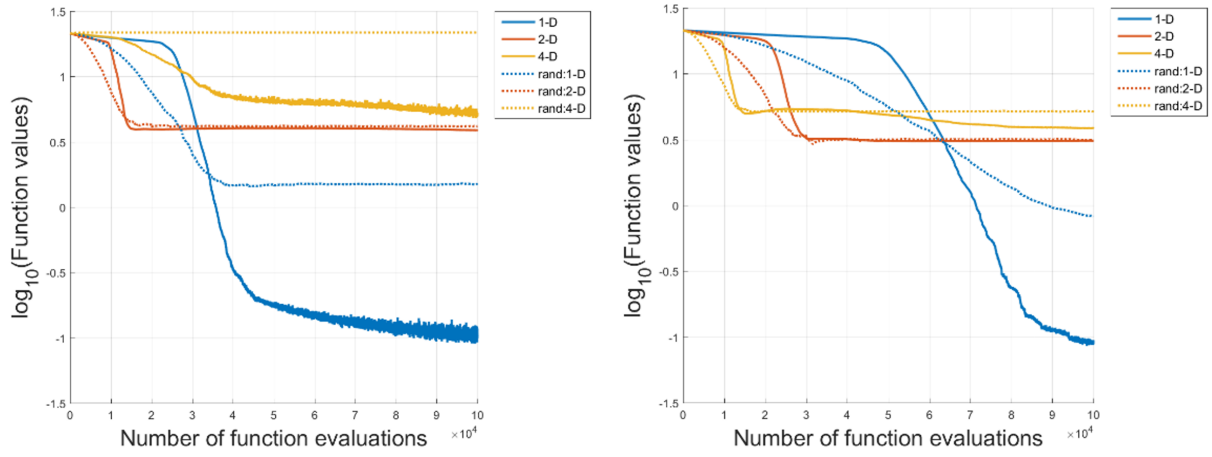
5.6.1 Surrogate Dimensionality Heuristic

This study proposes sub-dimensionality of surrogates as a new domain of research. This study clearly indicates that higher dimensional sub-surrogates offer better exploration and faster convergence but suffers from low sampling density. This can be alleviated by decreasing the sampling



(a) Ackley (b) Sum of squares

Figure 5.6: Decreasing sub-surrogate dimensions for (a) Ackley ($p = 16$) and (b) Sum of squares ($p = 64$) from 4-D to 2-D to 1-D.



(a) $p = 8$ (b) $p = 16$

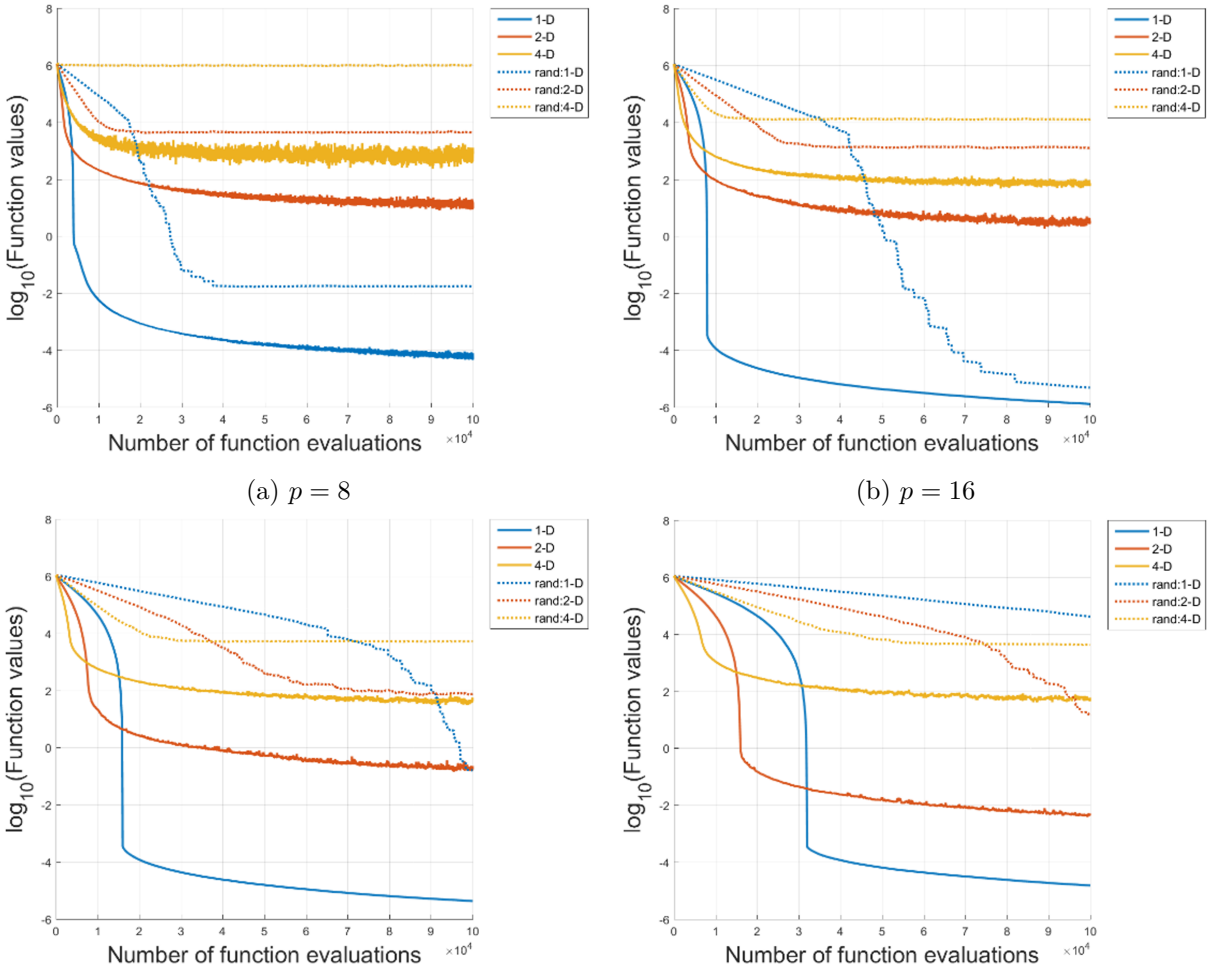
Figure 5.7: Ackley function sampled with (a) $p = 8$ and (b) $p = 16$ data points at each iteration. Sub-dimensional variable selection (1-D, 2-D and 4-D) is based on the gradient vector or randomly selected (rand:1D, rand:2D and rand:4D).

domain, increasing the number of samples, and as shown in the results of this study reducing the dimensionality of the domain over which the sub-dimensional surrogate is constructed.

Heuristics that identifies when to rather construct as a lower dimensional surrogate is very important. The benefits of such a heuristic are demonstrated in Figure 5.6(a) and (b) for the Ackley and Sum of squares test functions, where we start off initially with the 4-D sub-surrogate that is then reduced to 2-D and finally 1-D for a sampling density of 16-points per sub-dimensional surrogate.

5.6.2 Sampling Dimensions Heuristic or Strategy

Strategies to identify appropriate sub-dimensions for the construction of the surrogates also plays a critical role. In this study, we considered a basic and effectively greedy strategy, where we selected the partial derivatives of the gradient vector that is the highest magnitude. We demonstrate the importance of a selection heuristic and selection strategy by considering the random selection of a sub-space for the 1-D (denoted rand:1-D), 2-D (denoted rand:2-D) and 4-D (denoted rand:4-D) sub-surrogates for Ackley and Sum of squares which are depicted in Figures 5.7(a)-(b) – 5.8(a)-(d).



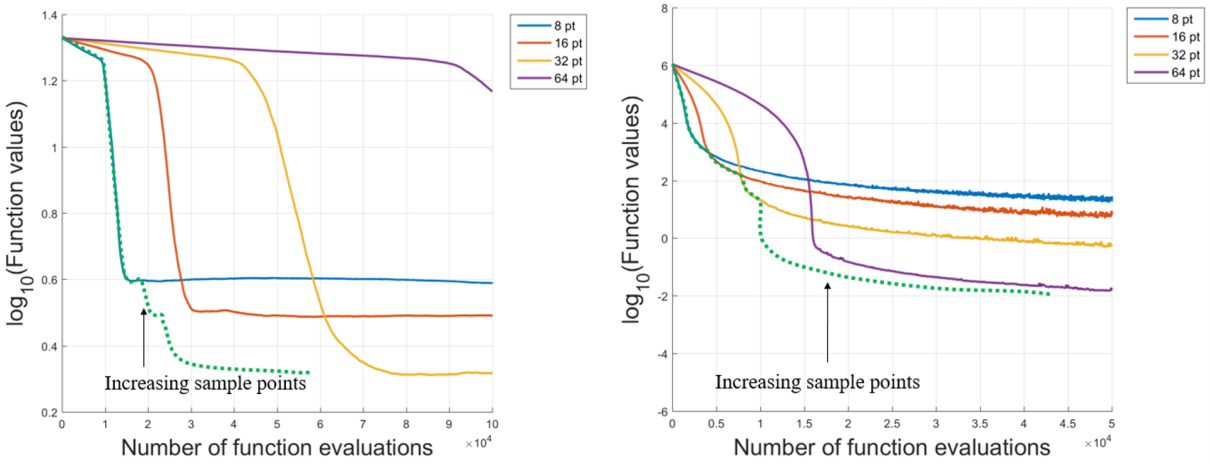
(a) $p = 8$

(b) $p = 16$

(c) $p = 32$

(d) $p = 64$

Figure 5.8: Sum of squares function sampled with (a) $p = 8$, (b) $p = 16$, (c) $p = 32$, and (d) $p = 64$ data points at each iteration. Sub-dimensional variable selection (1-D, 2-D, and 4-D) is based on the gradient vector or randomly selected (rand:1D, rand:2D and rand:4D).



(a) Ackley (b) Sum of squares

Figure 5.9: Increasing the number of sampling points for (a) Ackley and (b) Sum of squares for the 4-D sub-surrogate.

For the Ackley function, it is evident that the lower the sub-surrogate dimension the more sensitive the solution to variable interaction and the more care should be exercised to select sub-dimensions. In turn, the 2-D and 4-D sub-dimensional surrogates were less affected. In addition, it is clear that there is room for improvement in using the magnitude of gradient components for the 2-D and 4-D sub-dimensional surrogates, as the randomly selected sub-dimensions had better initial performance than the sub-dimensions selected on the gradient criteria on Ackley function.

That said for the Sum of squares function, the gradient component magnitude strategy significantly outperformed the randomly selected sub-dimensions for the 1-D, 2-D and 4-D sub-dimensional surrogates.

There is significant room for improvement on our investigated greedy gradient component magnitude strategy to identify appropriate sub-dimensions for future research.

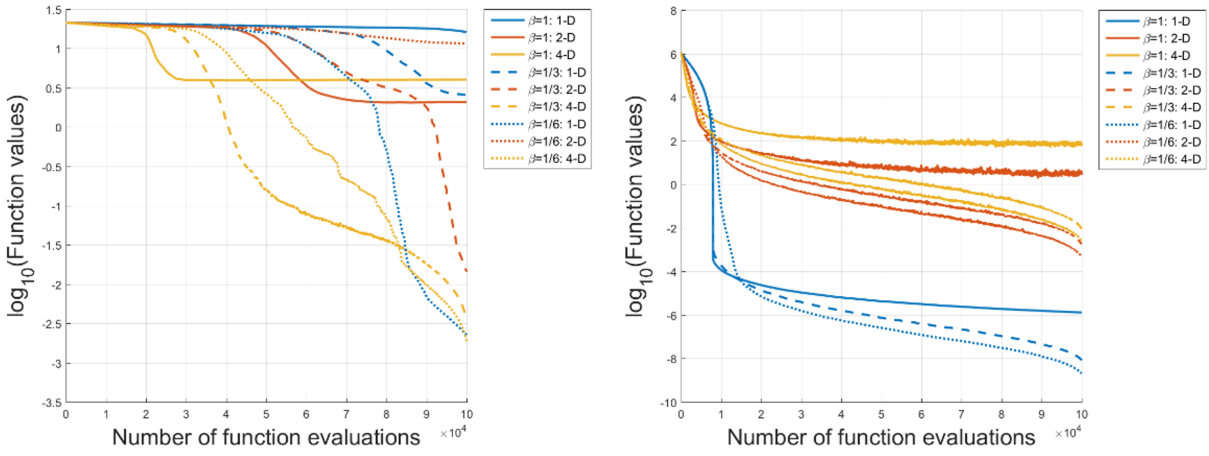
5.6.3 Sampling Quantity Heuristic

The number of sampling points can be increased e.g. doubled by going from 8 to 16 to 32 and 64 to obtain the performance depicted in Figure 5.9(a) and (b) for Ackley and Sum of squares. It is evident from Figure 5.6(a) and (b) that reducing the dimensionality seems to be a much more beneficial strategy to increase sampling density than increasing the number of points. That said, increasing the number of points allows for a finer grained control of the sampling density, i.e. a slight increase in the number of sampling points may supply enough information and allow for a useful higher dimensional surrogate to be constructed.

5.6.4 Sampling Domain Heuristics

Closely related to the sampling quantity is the sampling volume for a given sub-dimension. Reducing the sampling volume too aggressively results in a localized search early on, while reducing the sampling volume too slowly may waste significant computational resources. Again, the sampling allows for finer control of sampling density. In this study, we always constructed surrogates over the entire domain, however, Figure 5.10(a) and (b) depict the potential benefits of sampling an appropriate sub-domain. This was conducted by multiplying a dynamically changing scaling factor τ_n with the distances from the current point $\boldsymbol{x}_n$ to the boundaries of the surrogates. The expression for τ_n is given by

$$\tau_n = (\sqrt{v})^{-1}(1 - 10^{-10})\left(1 - \left(\frac{feval_n}{feval_{max}}\right)^{\beta}\right) + 10^{-10}. \tag{5.5}$$



(a) Ackley (b) Sum of squares

Figure 5.10: Decreasing the sampling volume for (a) Ackley ($p = 16$ points) and (b) Sum of squares ($p = 32$ points).

5.7 Neural Network Training

We now demonstrate that the behavior and performance observed on our test problems are related to actual machine learning problems. Towards this aim we consider three neural network architectures to solve the Iris dataset that has 4 input variables and 3 output variables for which the gradients were computed using backpropagation. We consider 3 problems of neural network (NN) with single hidden layer:

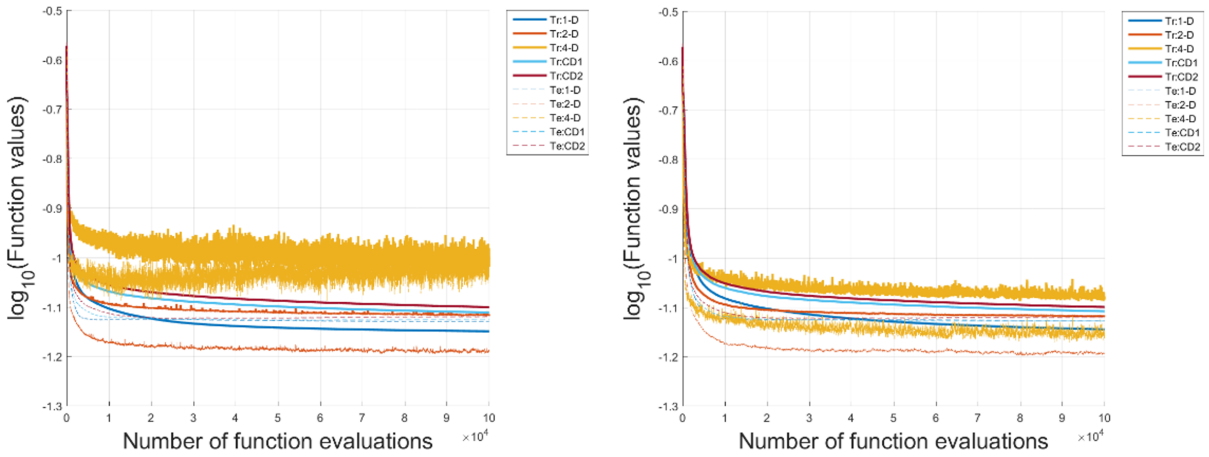
1. 1 node in the hidden layer resulted in a $(4 + 1)1 + (1 + 1)3 = 11$ variable problem;
2. 6 nodes in the hidden layer resulted in a $(4 + 1)6 + (6 + 1)3 = 51$ variable problem;
3. 63 nodes in the hidden layer resulted in a $(4 + 1)63 + (63 + 1)3 = 507$ variable problem.

All variables are initialized between -1 and 1. For training the network problems, a heuristic method is implemented such that the optimization range is not fixed between -1 and 1 but the range may slide dynamically along with the current solution $\boldsymbol{x}_n$ so that $\boldsymbol{x}_n$ always remains at the centre of the range. This method allows for the sampling density to be constant while the domain over which we sample may change.

The training errors are indicated with solid lines and the test errors are indicated by dotted lines for 11, 51 and 507-D in Figures 5.11-5.13.

The results for the 11-D problem are similar to the results obtained for the test problems. The multi-dimension surrogate search outperforms the univariate searches in terms of computational cost and generality as both the 2-D and 4-D achieve significantly improved test errors for $p = 32$ and 64 sampling points (see Figure 5.11(c) and (d)). In fact, 2-D achieved the best performance using $p = 8$ and $p = 16$ points, while 4-D saw significant improvement going from $p = 16$ to $p = 32$ points. Both training and test errors of the 4-D are significantly higher when lower number of sampling points were used (Figure 5.11(a)), however, as the number of points increase, the lower the minimum (Figure 5.11(d)).

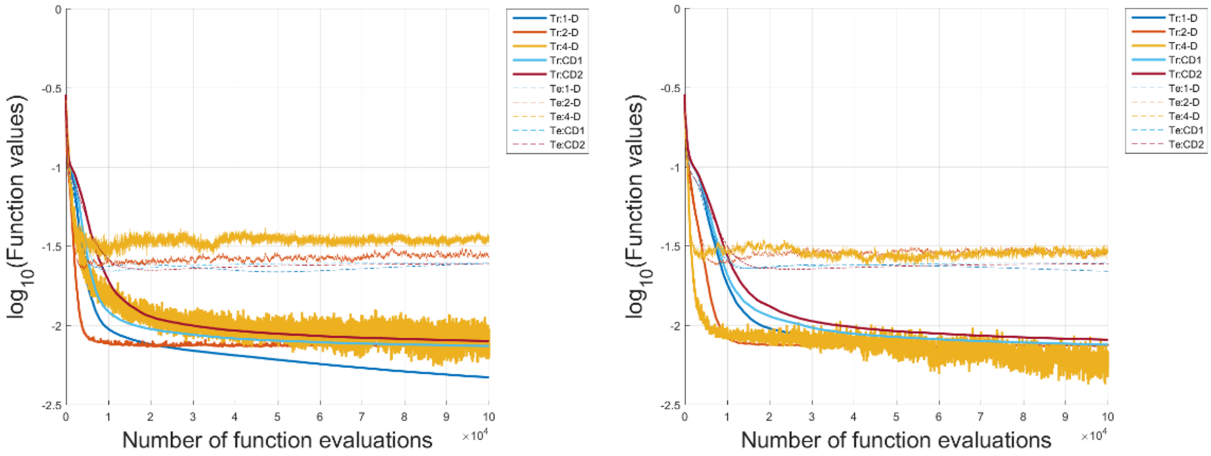
The trend between 51-D and 507-D are similar due to the simplicity of the problem due to overfitting (see Figures 5.12- 5.13). It needs to be emphasized that the test errors for both CD and 1-D sub-surrogate approaches are insensitive to the number of sampling points but for 2-D and 4-D sub-surrogate approaches, increasing the number of sampling points noticeably increases convergence rate with decreases in both training and test errors.



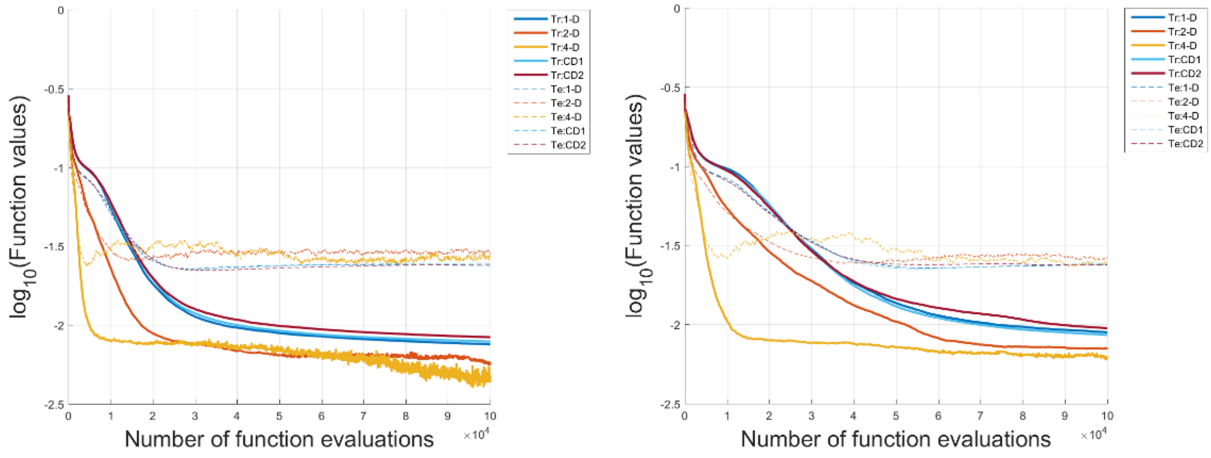
(a) $p = 8$ (b) $p = 16$

(c) $p = 32$ (d) $p = 64$

Figure 5.11: Training Iris data set with the gradient-based variable selection approach for 11-D.

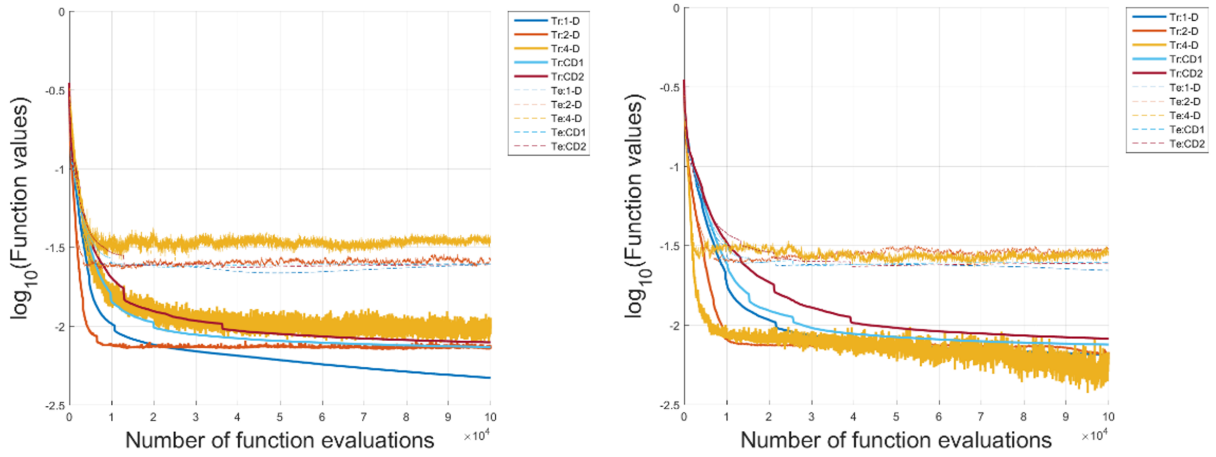


(a) $p = 8$ (b) $p = 16$



(c) $p = 32$ (d) $p = 64$

Figure 5.12: Training Iris data set with the gradient-based variable selection approach for 51-D.



(a) $p = 8$ (b) $p = 16$

(c) $p = 32$ (d) $p = 64$

Figure 5.13: Training Iris data set with the gradient-based variable selection approach for 507-D.

5.8 Conclusions

Currently surrogates are constructed for either 1-dimensions, i.e. line searches or full-dimensional, i.e. standard surrogates. This study proposes sub-dimensional surrogates, i.e. constructing surrogates in sub-dimensions between 1-dimension and the full-dimensions. We demonstrated that there is merit in exploring this research domain using merely a vanilla sub-dimensional surrogate strategy, without any heuristics, yes, we even omitted the obvious ones on purpose. This was done to ensure we investigate and learn from our investigation to gain further insight into this new research domain of surrogate construction, namely, sub-dimensional surrogates. We demonstrate that sub-dimensional surrogates are a viable approach to extend surrogate-based optimization to even higher dimensions on two test problems as formulated and randomly rotated to increase the variable interaction. Higher sub-dimensions allow for better initial improvement and exploration that are less sensitive to variable interactions, while lower sub-dimensions allow for better exploitation and are more sensitive to variable interactions. This study introduces two new domains for heuristics in surrogate construction namely sub-dimensions to control sampling density and heuristics to identify appropriate sub-dimensional spaces.

Based on our findings, we have identified the following areas of future research, which are to develop heuristics that:

1. Identify when to reduce the dimensionality of the sub-dimensional surrogate;
2. Identifying appropriate sub-dimensions;
3. Scale the number of samples and sampling domain (sub-space) given the sub-dimension.

We concluded our study by showing that the carefully selected test problems for our study are related to actual cost functions encountered in machine learning by training three neural nets on a well-known test problem in machine learning, namely the Iris data set.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The thesis searched for potential approaches for training deep neural network (DNN) architectures using 1-D approximation models and reduced-dimensional surrogate models because the computational cost of evaluating the function models grows every year. It is critical to keep exploring alternative ways to training those complex architectures.

The motivation for utilizing approximation and surrogate models for optimization in DNN is summarized to three points:

1. Efficiency for analyzing computationally expensive problems;
2. No need for figuring out the actual mathematical formulation for the black-box problems;
3. Transformation of discontinuous functions into smooth and continuous functions.

We began employing the 1-dimensional approximations as line searches because we may quickly build up the complexity of problems because it is the lowest dimensional model. The complexity includes the model errors manifested as biases and variances due to dynamic mini-batch sub-sampling. To compare the performance of each line search method against other learning rate strategies, we introduced a new relative robustness measure that considers both inferior and best performances of each strategy across different problems. This is in contrast to the traditional performance measure, which only analyzes the best performances. We can use the relative robustness measure for choosing benchmark algorithms for unseen problems.

The first approach of employing a 1-D approximation for optimization was minimization using the quadratic approximations. Although it may be the simplest model with the single minimum with limited flexibility, we considered all possible variations in the information required for the approximation. These included only function values, only directional derivatives, and the three more mixtures of both function values and directional derivatives, hence, five quadratic models in total.

Investigating the approximation errors of each model showed that the directional derivative data at the starting point (origin) of the model is critical for reducing the variance in the errors, and having a large amount of data does not necessarily decrease the model errors. This was a crucial finding for line searches since it aims to find the local minima using minimization for this approximation type.

Next, we enforced conservatism to the best performing quadratic approximation, ensuring convergence with provided proof using a bracketing strategy at the cost of performance and convergence rate. This allowed us to quantify the sacrifice in convergence rate when increasing the robustness. The chosen model for this development was the gradient-only surrogate (GOS) which requires two directional derivative points. The extended algorithm called Gradient-Only Approximation Line Search (GOALS) consecutively constructs GOS along descent directions to search for the stochastic non-negative gradient projection points (SNN-GPPs) until the curvature condition satisfies.

We conducted several hyperparameter studies on GOALS. The hyperparameters include the degrees of overshooting and undershooting, the initial guess, and whether to use the learning rate in the previous iteration for the next. Various hyperparameters allow for examining how they affect the performance of training DNN. The most robust initial guess at each iteration was not using the previous resultant learning rate but the inverse of the norm of the direction vector for each optimizer we employed. GOALS often does not allow to be as aggressive as GOS in training with convergence criteria. Still, GOALS ranked second for both training and test accuracies out of ten other strategies based on the relative robustness measure, led by GOS, for a shallow network problem, and GOALS outperformed GOS for deeper network problems. This showed deterministic approximation usage is a potential area for line searches and future researches.

Line searches optimize the learning rate to satisfy its specified cost functions for a descent direction. Therefore, it often requires more than one function evaluation per iteration. This led to the exploration of the bound approximation approach for line search using Bayesian classification. Our Gradient-Only Bayesian Classification Line Search (GOCLS) constructs two bound approximations of each directional derivative sign using historical data to assume that the location of the current SNN-GPPs is a function of previous iterations. Hence, GOCLS only requires a single gradient computation per iteration, even as a line search.

GOCLS is efficient in computational cost and ranks fourth for training and test accuracies among the seven state-of-the-art strategies, based on the relative robustness measures. These results showed the Bayesian classification-based approach a clear research potential as a line search. GOCLS is controlled by an intuitive hyperparameter called the target probability. This regulates how often the resultant learning would observe a certain sign of directional derivative, which means the user may control the degrees of overshooting and undershooting. The hyperparameter study for choosing the value showed that for all optimizers, including SGD, RMSProp, and Adam, the overshooting improves convergence rates in training.

The last chapter about the sub-dimensional surrogate model approach was an introductory chapter to a novel multi-dimensional selected variable model for training DNNs. We did not restrict the dimensionality of the approximation approach to one. However, We attempted to overcome the "curse of dimensionality" for the high-dimensional problem by employing a still small but multi-dimensional approach to better comprehend the actual problem's complexity.

To concentrate on the subsurrogate models' behavior, we kept the test problem as small as possible and utilized full-batch for clearing discontinuity. This does not mean that the model is oversimplified. The subsurrogate model was constructed using the cubic radial basis function with extra linear polynomial function to avoid under-rank issues.

As a crude subsurrogate model, several hyperparameters were explored, including the appropriate dimension of the model, the choice of descent directions, the number of data points, and the volume shrinkage rate for faster convergence. The approach tested on various high-dimensional problems showed that we maintain rapid convergence in optimization when the dimensionality of the subsurrogate reduces and the number of data points increases during training. Although the subsurrogate model is still at a crude stage, the study shows that it is worth investigating further for training DNNs as it is not as computationally expensive as a full-dimensional surrogate. It expresses much more details than a one-dimensional approximation, line searches. Approximation and surrogate models are powerful tools in engineering and sciences with the major downside of difficulties in high-dimensions. The thesis illustrated that it could overcome these challenges with lower-dimensional models, and we hope that this work sets a foundation for further researches.

6.2 Future work

As the future work for the deterministic function approximation line searches, we explore higher dimensionalities of approximations to express more nonlinearity in the highly nonlinear problems. This would require a trade-off between reducing the number of function evaluations

required and reducing the approximation errors. The computational cost of higher dimensional models would be compensated with the usage of the immediate accept conditions (IAC). One could also adaptively change the dimensionality of approximations over the number of iterations.

As the future work of the stochastic bound approximation line searches, one could explore various probability density functions to capture directional derivative signs' distributions better. The target probability of observing certain signs does not need to be fixed as we experimented. This means that the target probability hyperparameter could vary over training DNNs. To prevent the learning rates from increasing explodingly, one could attempt lowering the hyperparameter over the training. We also explore the number of historical data points to consider for constructing bound approximation. This hyperparameter could increase as approaching the end of training for steadier changes in learning rates.

As future work of the last chapter about subsurrogate models, we could develop a model in which each weight consists of linear or nonlinear combinations of multiple variables in the actual problem. In this way, the subsurrogate model would control more variables simultaneously, enhancing training speed and requiring fewer dimensions. However, choosing such combined directions may require expensive computations. Hence, it is critical to efficiently finding the causalities between variables. If one designs a model too complex, it requires more function evaluated points to regress on. Therefore, there needs to be a good balance between the resolution of the model and computational cost. Although the thesis only explored the function-value-based subsurrogate models, Snyman and Wilke [2018] introduces a gradient-only surrogate model for discontinuous functions. Employing this to our surrogate approach may be a good direction for the subsequent research to resolve the discontinuities in DNNs due to dynamic mini-batch sub-sampling.

Appendix A

Appendix

A.1 Pseudocode for various approximations

Algorithm 6: `StepSizeFGF`

Input: α_1, $\tilde{f}_0$, $\tilde{f}_1$, $\tilde{f}'_0$, ε
Output: α^*

$\alpha^* = \alpha_1$
Define a matrix $\boldsymbol{A}_2$ and a vector $\boldsymbol{b}_2$ from (2.14)
if $rank(\boldsymbol{A}) = 3$ **then**
    Solve for the constants $\boldsymbol{k} = \boldsymbol{A}^{-1}\boldsymbol{b}$ from (2.10)
    if $k_1 > \varepsilon$ **then**
        $\alpha^* = \max(\alpha_{min}, \min(-k_2/(2k_1), \alpha_{max}))$

Algorithm 7: `StepSizeFFG`

Input: α_1, $\tilde{f}_0$, $\tilde{f}_1$, $\tilde{f}'_1$, ε
Output: α^*

$\alpha^* = \alpha_1$
Define a matrix $\boldsymbol{A}_3$ and a vector $\boldsymbol{b}_3$ from (2.15)
if $rank(\boldsymbol{A}) = 3$ **then**
    Solve for the constants $\boldsymbol{k} = \boldsymbol{A}^{-1}\boldsymbol{b}$ from (2.10)
    if $k_1 > \varepsilon$ **then**
        $\alpha^* = \max(\alpha_{min}, \min(-k_2/(2k_1), \alpha_{max}))$

Algorithm 8: `StepSizeFGFG`

Input: α_1, $\tilde{f}_0$, $\tilde{f}_1$, $\tilde{f}'_0$, $\tilde{f}'_1$, ε
Output: α^*

$\alpha^* = \alpha_1$
Define a matrix $\boldsymbol{A}_4$ and a vector $\boldsymbol{b}_4$ from (2.16)
if $rank(\boldsymbol{A}) = 3$ **then**
    Solve for the constants $\boldsymbol{k} = \boldsymbol{A}^{-1}\boldsymbol{b}$ from (2.10)
    if $k_1 > \varepsilon$ **then**
        $\alpha^* = \max(\alpha_{min}, \min(-k_2/(2k_1), \alpha_{max}))$

Algorithm 9: `StepSizeGG`

Input: α_1, $\tilde{f}_0'$, $\tilde{f}_1'$, ε
Output: α^*

$\alpha^* = \alpha_1$
Define a matrix $\boldsymbol{A}_5$ and a vector $\boldsymbol{b}_5$ from (2.17)
if $rank(\boldsymbol{A}) = 2$ **then**
    Solve for the constants $\boldsymbol{k} = \boldsymbol{A}^{-1}\boldsymbol{b}$ from (2.12)
    if $k_1 > \varepsilon$ **then**
        $\alpha^* = \max(\alpha_{min}, \min(-k_2/(2k_1), \alpha_{max}))$

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012. ISBN 978-3-642-35288-1.

Yoshua Bengio, Ian Goodfellow, and Aaron Courville. *Deep learning*, volume 1. MIT press Massachusetts, USA:, 2017.

El-houcine Bergou, Youssef Diouane, Vladimir Kunc, Vyacheslav Kungurtsev, and Clément W Royer. A subsampling line-search method with second-order results. *arXiv preprint arXiv:1810.07211*, 2018.

Dimitri P Bertsekas. Nonlinear programming. *Journal of the Operational Research Society*, 48 (3):334–334, 1997.

Atharv Bhosekar and Marianthi Ierapetritou. Advances in surrogate based modeling, feasibility analysis, and optimization: A review. *Computers & Chemical Engineering*, 108:250–267, 2018.

Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

Raghu Bollapragada, Richard Byrd, and Jorge Nocedal. Adaptive sampling strategies for stochastic optimization. *SIAM Journal on Optimization*, 28(4):3312–3343, 2018.

Antoine Bordes, Léon Bottou, and Patrick Gallinari. Sgd-qn: Careful quasi-newton stochastic gradient descent. *Journal of Machine Learning Research*, 10:1737–1754, 2009.

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

Leon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization Methods for Machine Learning. *SIAM Review*, 60(2):223–311, 2018. ISSN 0036-1445. doi: 10.1137/16M1080173.

Wadii Boulila, Maha Driss, Mohamed Al-Sarem, Faisal Saeed, and Moez Krichen. Weight initialization techniques for deep learning algorithms in remote sensing: Recent trends and future perspectives. *arXiv preprint arXiv:2102.07004*, 2021.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh,

Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf`.

R. H. Byrd, G. M. Chin, J. Nocedal, and Y. Wu. Sample Size Selection in Optimization Methods for Machine Learning. *Mathematical Programming*, 134(1):127–155, August 2012a. ISSN 0025-5610. doi: 10.1007/s10107-012-0572-5.

Richard H Byrd, Gillian M Chin, Will Neveitt, and Jorge Nocedal. On the use of stochastic hessian information in optimization methods for machine learning. *SIAM Journal on Optimization*, 21(3):977–995, 2011.

Richard H Byrd, Gillian M Chin, Jorge Nocedal, and Yuchen Wu. Sample size selection in optimization methods for machine learning. *Mathematical programming*, 134(1):127–155, 2012b.

Adrian A Canutescu and Roland L Dunbrack Jr. Cyclic coordinate descent: A robotics algorithm for protein loop closure. *Protein science*, 12(5):963–972, 2003.

Younghwan Chae and Daniel N Wilke. Empirical study towards understanding line search approximations for training neural networks. *arXiv preprint arXiv:1909.06893*, 2019a.

Younghwan Chae and Daniel N Wilke. Sub-dimensional surrogates to solve high dimensional optimization problems in machine learning. In Sergey Yurish, editor, *Advances in Artificial Intelligence: Reviews*, chapter 2, pages 59–83. IFSA Publishing, 2019b.

Younghwan Chae and Daniel N Wilke. Empirical study towards understanding line search approximations for training neural networks. *arXiv preprint arXiv:1909.06893*, 2019c.

Younghwan Chae, Daniel N Wilke, and Dominic Kafka. GOALS: Gradient-only approximations for line searches towards robust and consistent training of deep neural networks. *arXiv preprint arXiv:2105.10915*, 2021.

Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Lin. Coordinate descent method for large-scale l2-loss linear support vector machines. *Journal of Machine Learning Research*, 9(7), 2008.

Liming Chen, Haobo Qiu, Liang Gao, Chen Jiang, and Zan Yang. A screening-based gradient-enhanced kriging modeling method for high-dimensional problems. *Applied Mathematical Modelling*, 69:15–31, 2019.

Dominik Csiba and Peter Richtárik. Importance sampling for minibatches. *The Journal of Machine Learning Research*, 19(1):962–982, 2018.

Timothy Dozat. Incorporating nesterov momentum into adam. 2016.

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

Michael P Friedlander and Mark Schmidt. Hybrid deterministic-stochastic methods for data fitting. *SIAM Journal on Scientific Computing*, 34(3):A1380–A1405, 2012.

Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2008.

Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning (Adaptive Computation and Machine Learning series)*. The MIT Press, 2016. ISBN 0262035618.

Ian J Goodfellow, Oriol Vinyals, and Andrew M Saxe. Qualitatively characterizing neural network optimization problems. *arXiv preprint arXiv:1412.6544*, 2014.

Rajesh Kumar Gupta. *Numerical Methods: Fundamentals and Applications*. Cambridge University Press, 2019.

Zhong-Hua Han, Stefan Görtz, and Ralf Zimmermann. Improving variable-fidelity surrogate modeling via gradient-enhanced kriging and a generalized hybrid bridge function. *Aerospace Science and technology*, 25(1):177–189, 2013.

Zhong-Hua Han, Yu Zhang, Chen-Xing Song, and Ke-Shi Zhang. Weighted gradient-enhanced kriging for high-dimensional surrogate modeling and design optimization. *Aiaa Journal*, 55 (12):4330–4346, 2017.

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*, pages 770–778, 2016.

Ilija Ilievski, Taimoor Akhtar, Jiashi Feng, and Christine Shoemaker. Efficient hyperparameter optimization for deep learning algorithms using deterministic rbf surrogates. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. *Advances in neural information processing systems*, 26:315–323, 2013.

D. Kafka and D. N. Wilke. Resolving Learning Rates Adaptively by locating Stochastic Non-Negative Associated Gradient Projection Points using Line Searches. Unpublished: In review at the Journal of Global Optimization, 7 2019a.

Dominic Kafka and Daniel Wilke. Gradient-only line searches: An alternative to probabilistic line searches. *arXiv preprint arXiv:1903.09383*, 2019b.

Dominic Kafka and Daniel N Wilke. Resolving learning rates adaptively by locating stochastic non-negative associated gradient projection points using line searches. *Journal of Global Optimization*, 79(1):111–152, 2021.

Kyeonghwan Kang, Ikjin Lee, and Donghyun Kim. Efficient metamodeling strategy using multivariate linear interpolation for high dimensional problems. In *World Congress of Structural and Multidisciplinary Optimisation*, pages 234–241. Springer, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. *Science Department, University of Toronto, Tech.*, 2009. ISSN 1098-6596. doi: 10.1.1.222.9220.

Martin Kubicek, Edmondo Minisci, and Marco Cisternino. High dimensional sensitivity analysis using surrogate modeling and high dimensional model representation. *International Journal for Uncertainty Quantification*, 5(5), 2015.

Jan Kukačka, Vladimir Golkov, and Daniel Cremers. Regularization for deep learning: A taxonomy. *arXiv preprint arXiv:1710.10686*, 2017.

Vyacheslav Kungurtsev and Tomas Pevny. Algorithms for solving optimization problems arising from deep neural net models: smooth problems. *arXiv preprint arXiv:1807.00172*, 2018.

Luc Laurent, Rodolphe Le Riche, Bruno Soulier, and Pierre-Alain Boucard. An overview of gradient-enhanced metamodels with applications. *Archives of Computational Methods in Engineering*, 26(1):61–106, 2019.

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

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *arXiv preprint arXiv:1712.09913*, 2017.

K. Liu. 95.16% on CIFAR10 with PyTorch. `https://github.com/kuangliu/pytorch-cifar`, 2020.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, aug 2017a. URL `https://arxiv.org/abs/1608.03983`.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017b.

Zhi-Quan Luo and Paul Tseng. A coordinate gradient descent method for nonsmooth separable minimization. *Journal of optimization theory and applications*, 72(1), 2002.

Aleksandr Mikhailovich Lyapunov. The general problem of the stability of motion. *International journal of control*, 55(3):531–534, 1992.

Sedigheh Mahdavi, Mohammad Ebrahim Shiri, and Shahryar Rahnamayan. Cooperative co-evolution with a new decomposition method for large-scale optimization. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 1285–1292. IEEE, 2014.

Maren Mahsereci and Philipp Hennig. Probabilistic line searches for stochastic optimization. *arXiv preprint arXiv:1502.02846*, 2015.

Maren Mahsereci and Philipp Hennig. Probabilistic line searches for stochastic optimization. *The Journal of Machine Learning Research*, 18(1):4262–4320, 2017.

Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*, 2018.

MATLAB. *version 8.6.0.267246*. The MathWorks Inc., Natick, Massachusetts, 2015.

Michael D McKay, Richard J Beckman, and William J Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 42(1):55–61, 2000.

Maximus Mutschler and Andreas Zell. Parabolic approximation line search: An efficient and effective line search approach for DNNs. *arXiv preprint arXiv:1903.11991*, 2019.

John A Nelder and Roger Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

Yu Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Mitchell A Potter and Kenneth A De Jong. A cooperative coevolutionary approach to function optimization. In *International Conference on Parallel Problem Solving from Nature*, pages 249–257. Springer, 1994.

Michael JD Powell. The theory of radial basis function approximation in 1990. *Advances in numerical analysis*, pages 105–210, 1992.

Mike JD Powell. Five lectures on radial basis functions. *Informatics and Mathematical Modelling, Technical University of Denmark, DTU*, 2005.

Nestor V Queipo, Raphael T Haftka, Wei Shyy, Tushar Goel, Rajkumar Vaidyanathan, and P Kevin Tucker. Surrogate-based analysis and optimization. *Progress in aerospace sciences*, 41(1):1–28, 2005.

Rommel G Regis. Constrained optimization by radial basis function interpolation for high-dimensional expensive black-box problems with infeasible initial points. *Engineering Optimization*, 46(2):218–243, 2014.

Rommel G Regis and Christine A Shoemaker. Combining radial basis function surrogates and dynamic coordinate search in high-dimensional expensive black-box optimization. *Engineering Optimization*, 45(5):529–555, 2013.

Peter Richtárik and Martin Takáč. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming*, 144(1):1–38, 2014.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Ken Sauer and Charles Bouman. A local update strategy for iterative reconstruction from projections. *IEEE Transactions on Signal Processing*, 41(2):534–548, 1993.

Songqing Shan and G Gary Wang. Survey of modeling and optimization strategies to solve high-dimensional design problems with computationally-expensive black-box functions. *Structural and multidisciplinary optimization*, 41(2):219–241, 2010.

Fanhua Shang, Kaiwen Zhou, Hongying Liu, James Cheng, Ivor W Tsang, Lijun Zhang, Dacheng Tao, and Licheng Jiao. Vr-sgd: A simple stochastic variance reduction method for machine learning. *IEEE Transactions on Knowledge and Data Engineering*, 32(1):188–202, 2018.

Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.

Nishant Shukla and Kenneth Fricklas. *Machine learning with TensorFlow*. Manning Greenwich, 2018.

Wei Shyy, Young-Chang Cho, Wenbo Du, Amit Gupta, Chien-Chou Tseng, and Ann Marie Sastry. Surrogate-based modeling and dimension reduction techniques for multi-scale mechanics problems. *Acta Mechanica Sinica*, 27(6):845–865, 2011.

Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pages 464–472. IEEE, 2017.

Jan A.. Snyman and Daniel N.. Wilke. *Practical Mathematical Optimization: Basic Optimization Theory and Gradient-based Algorithms*. Springer., 2018.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.

Chaoli Sun, Yaochu Jin, Ran Cheng, Jinliang Ding, and Jianchao Zeng. Surrogate-assisted cooperative swarm optimization of high-dimensional expensive problems. *IEEE Transactions on Evolutionary Computation*, 21(4):644–660, 2017.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2): 26–31, 2012.

Bryan A Tolson and Christine A Shoemaker. Dynamically dimensioned search algorithm for computationally efficient watershed model calibration. *Water Resources Research*, 43(1), 2007.

Ky Khac Vu, Claudia d'Ambrosio, Youssef Hamadi, and Leo Liberti. Surrogate-based methods for black-box optimization. *International Transactions in Operational Research*, 24(3):393–424, 2017.

Chong Wang, Xi Chen, Alex Smola, and Eric P. Xing. Variance reduction for stochastic gradient optimization. In C J C Burges, L Bottou, M Welling, Z Ghahramani, and K Q Weinberger, editors, *Advances in Neural Information Processing Systems*, pages 181–189. Curran Associates, Inc., 2013. URL `http://papers.nips.cc/paper/5034-variance-reduction-for-stochastic-gradient-optimization.pdf`.

Bernhard Werth, Erik Pitzer, and Michael Affenzeller. Enabling high-dimensional surrogate-assisted optimization by using sliding windows. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1630–1637, 2017.

Daniel N Wilke. How to get rid of discontinuities when constructing surrogates from piecewise discontinuous functions. In *Proceedings of the 7th International Conference on Discrete Element Methods. Springer*, 2016.

Daniel Nicolas Wilke, Schalk Kok, Johannes Arnoldus Snyman, and Albert A Groenwold. Gradient-only approaches to avoid spurious local minima in unconstrained optimization. *Optimization and Engineering*, 14(2):275–304, 2013.

Adrian G. Wills and Thomas B. Schön. On the construction of probabilistic Newton-type algorithms. In *2017 IEEE 56th Annual Conference on Decision and Control, CDC 2017*, volume 2018-Janua, pages 6499–6504. IEEE, 2018. ISBN 9781509028733. doi: 10.1109/CDC.2017.8264638.

Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, volume 2017-December, pages 4149–4159, 2017.

Philip Wolfe. Convergence conditions for ascent methods. *SIAM review*, 11(2):226–235, 1969.

Philip Wolfe. Convergence conditions for ascent methods. ii: Some corrections. *SIAM review*, 13(2):185–188, 1971.

Stephen J Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

Lin Xiao and Tong Zhang. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4):2057–2075, 2014.

Zhenyu Yang, Ke Tang, and Xin Yao. Large scale evolutionary optimization using cooperative coevolution. *Information sciences*, 178(15):2985–2999, 2008a.

Zhenyu Yang, Ke Tang, and Xin Yao. Multilevel cooperative coevolution for large scale optimization. In *2008 IEEE congress on evolutionary computation (IEEE World Congress on Computational Intelligence)*, pages 1663–1670. IEEE, 2008b.

Rahul Yedida, Snehanshu Saha, and Tejas Prashanth. Lipschitzlr: Using theoretically computed adaptive learning rates for fast convergence. *Applied Intelligence*, 51(3):1460–1478, 2021.

Matthew D. Zeiler. ADADELTA: An Adaptive Learning Rate Method. dec 2012a. URL `http://arxiv.org/abs/1212.5701`.

Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012b.

Pan Zhou, Jiashi Feng, Chao Ma, Caiming Xiong, Steven HOI, et al. Towards theoretically understanding why sgd generalizes better than adam in deep learning. *arXiv preprint arXiv:2010.05627*, 2020.

Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, 67(2):301–320, 2005.