Automated learning rates in machine learning for dynamic mini-batch sub-sampled losses

by

Dominic Kafka

This thisis 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

2020

To the pursuit of progress..

"We never get there, we only get better." - Daniel N. Wilke
A gradient-only projection into the philosophical space.

Abstract

Title: Automated learning rates in machine learning for dynamic mini-batch sub-sampled losses

Author: Dominic Kafka

Supervisor: Daniel N. Wilke

Learning rate schedule parameters remain some of the most sensitive hyperparameters in machine learning, as well as being challenging to resolve, in particular when mini-batch sub-sampling is considered. Mini-batch sub-sampling (MBSS) can be conducted in a number of ways, each with their own implications on the smoothness and continuity of the underlying loss function. In this study, dynamic MBSS, often applied in approximate optimization, is considered for neural network training. For dynamic MBSS, the mini-batch is updated for every function and gradient evaluation of the loss and gradient functions. The implication is that the sampling error between mini-batches changes abruptly, resulting in non-smooth and discontinuous loss functions. This study proposes an approach to automatically resolve learning rates for dynamic MBSS loss functions using gradient-only line searches (GOLS) over fifteen orders of magnitude. A systematic study is performed, which investigates the characteristics and the influence of training algorithms, neural network architectures and activation functions on the ability of GOLS to resolve learning rates. GOLS are shown to compare favourably against the state-of-the-art probabilistic line search for dynamic MBSS loss functions. Matlab and PyTorch 1.0 implementations of GOLS are available for both practical training of neural networks as well as a research tool to investigate dynamic MBSS loss functions.

Acknowledgements

First and foremost I would like express my gratitude towards Prof. Daniel N. Wilke for his supervision over the duration of my post-graduate studies. His guidance ensured that I remained outside of my comfort zone and in the domain of maximal learning, while providing a solid soundboard of clarity while attempting to extract concepts out of the murky ether of the unknown. It is generally a shame, that I rarely appreciate the growing that occurred during various stages of my life, and the persons who contributed towards this it, until a particular scenario lays it bare. Prof, it has been a privilege to be one of your students, and although I am already extremely grateful for your guidance towards becoming an increasingly conceptual thinker, I am sure that with time more hidden learned traits will be uncovered. I humbly thank you in advance for this.

I also would like to tip my hat to Younghwan Chae for his contribution towards exploring new ideas and concepts. As my "office-inmate", you were the first person to be exposed to new ponderings and assist in separating the obvious chaff from the wheat. Thank you for that, your feedback provided a quick feed-back loop, which prevented me from becoming lost in spurious thought experiments.

I would also like to thank 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, which supported this work.

To my family, friends, and Leigh-Anne: I could write another 100+ page document on how much your support has meant to me over the years. Also, it would take just as many pages to list all the names that have contributed in this manner. However, sometimes I find beauty in the things that are implied and left undefined.. because how does one put unconditional love into words? I certainly cannot. It turns out, that it takes great effort to put abstract ideas into words that are precise enough to capture their full complexity. As Prof. Wilke has often mentioned, we are humble Engineers, not Rudyard Kipling. My words would just fail. However, I will say this: Thank you, with all that I am, because the sum of all your contributions makes up the biggest part of me. Over the integral of time, the little encouraging words here, the warm meal there, and the great conversations everywhere have all contributed to who I am, where I am and what I have achieved. Thank you dearly.

Contents

List of Abbreviations **15**

List of Symbols **16**

1 Overview **17**

2 Traversing the noise of dynamic mini-batch sub-sampled loss functions: A visual guide **20**

2.1 Introduction: The stochastic neural network optimization problem 20

2.1.1 Our contribution 25

2.2 Application of concepts to a practical neural network problem 26

2.2.1 Full combinatorial static and dynamic mini-batch sub-sampling 26

2.2.2 Variance properties of function value and directional derivative information in the MSE loss using dynamic MBSS 28

2.2.3 The influence of variance on minimum and SNN-GPP optimality criteria 31

2.2.4 Optimality criteria along 1-D search directions 33

2.3 Alternative Activation Functions 35

2.3.1 Global activation function characteristics in random directions over [-20,20] domains in weight space 36

2.3.2 Local activation function characteristics in descent directions over [-2,2] domains in weight space 37

2.4 Sensitivity of optimum formulations to mini-batch sample size 39

2.5 Conclusion 40

3 Resolving learning rates adaptively by locating SNN-GPPs using line searches **42**

3.1 Introduction 42

3.1.1 Loss Functions and Gradients 42

3.1.2 Optimization Formulations: Full-Batch Sampling 44

3.1.3 Optimization Formulations: Static Mini-Batch Sub-Sampling 46

3.1.4 Optimization Formulations: Dynamic Mini-Batch Sub-Sampling 47

3.1.5 Related Work 51

3.1.6 Contribution 52

3.2 Method 53

3.2.1 Stochastic Gradient Descent with Line Search Strategies 53

3.2.2 Exact line searches: Function Value based Golden Section (GS) and Gradient-only Line Search with Bisection (GOLS-B) 54

3.2.3 Inexact line search: Function value based Armijo's rule (ARLS) and Gradient-Only Line Search that is Inexact (GOLS-I) 55

3.2.4 Theoretical Basis 57

3.3 Numerical Studies 58

3.3.1 Diverse Foundational Classification Problems 59

3.3.2 Variational Autoencoder Training 61

3.3.3 Comparison between static and dynamic mini-batch sub-sampling 61

3.3.4 Naming convention 61

3.4 Results 62
3.4.1 Averaged results 62
3.4.2 Examples of individual best performances for different methods 63
3.4.3 Comparing step size characteristics of the Glass1 and Cancer1 datasets . 65
3.4.4 Analysis of iterative performance of line search methods 66
3.4.5 Variational Autoencoder Training Investigation 67
3.4.6 A demonstration of the merits of dynamic mini-batch sub-sampling . . . 71
3.5 Conclusion 73

4 Gradient-only line searches: An Alternative to Probabilistic Line Searches 75
4.1 Introduction 75
4.1.1 Our Contribution 77
4.2 Loss Functions and means of locating optima 78
4.3 Empirical evidence of improved localization of optima with SNN-GPPs over minimizers 82
4.4 Algorithmic Details 83
4.4.1 Proof of Global Convergence for Full-Batch Sampling 86
4.4.2 Proof of Global Convergence for Dynamic Mini-Batch Sub-Sampling . . . 87
4.5 Numerical Studies 88
4.6 Results 90
4.6.1 The Breast Cancer Wisconsin Diagnostic (BCWD) Dataset with Logistic Regression, NetPI and NetPII 90
4.6.2 The MNIST Dataset with NetI and NetII 93
4.6.3 The CIFAR10 dataset with NetI and NetII 95
4.7 Differences to the original PrLS study 97
4.8 PrLS and GOLS-I: Line searches, not update rules 98
4.9 Conclusion 100

5 Gradient-only line searches to automatically determine learning rates in stochastic training algorithms 102
5.1 Introduction 102
5.2 Connections: Approaches in advancing neural network training 103
5.2.1 Modifying loss function landscapes 104
5.2.2 Improving optimizers 105
5.3 Our contribution 106
5.4 A summary of gradient-only line search concepts 107
5.5 Training algorithms and constructing search directions 111
5.6 Numerical studies 112
5.6.1 Single hidden layer feedforward neural networks 112
5.6.2 Multiple hidden layer neural networks 118
5.6.3 CIFAR10 with ResNet18 120
5.7 Conclusion 123

6 The compatibility of Gradient-only line searches with various activation functions 124
6.1 Introduction 124
6.2 Connections: Gradient-only line searches and activation functions 127
6.3 Contribution 130
6.4 Empirical Study 131
6.5 Results 132
6.5.1 Foundational problems 133
6.5.2 A closer look at ReLU loss functions using GOLS-B SGD 134
6.5.3 MNIST with NetII 137
6.5.4 CIFAR10 with ResNet18 140

6.6 Conclusion 142

7 Conclusions and future work 144

7.1 Recommendations for future work 145

A Appendix 147

A.1 Figures of Additional Activation Functions from Visual Investigations 147

A.1.1 Relating to Global Activation Function Characteristics in Random Directions over [-20,20] Domains in Weight Space 148

A.1.2 Relating to Local Activation Function Characteristics in Descent Directions over [-2,2] Domains in Weight Space 153

A.2 Artificial Neural Networks 159

A.3 Pseudo code for Gradient-Only Line Searches 160

A.3.1 Exact Line Search: Gradient-Only Line Search with Bisection (GOLS-B) 160

A.3.2 Inexact Line Search: Gradient-Only Line Search that is Inexact (GOLS-I) 161

A.3.3 Inexact Line Search: Gradient-Only Line Search Maximizing step size (GOLS-Max) 162

A.3.4 Inexact Line Search: Gradient-Only Line Search with Backtracking (GOLS-Back) 163

A.4 Line Search Adapted optimization algorithms 163

A.4.1 Line Search Stochastic Gradient Descent (LS-SGD) 163

A.4.2 Line Search Stochastic Gradient Descent with Momentum (LS-SGDM) . . 164

A.4.3 Line Search Nesterov Accelerated Gradient Descent (LS-NAG) 164

A.4.4 Line Search Adagrad 164

A.4.5 Line Search Adadelta 165

A.4.6 Line Search Adam 165

A.5 Heuristics for determining the number of hidden units 166

List of Figures

2.1 Potential gradient descent (GD) updates in a neural network loss function for the Iris [Fisher, 1936] classification problem. 21

2.2 Conceptual depiction of static and dynamic mini-batch sub-sampled loss functions. (a,b) The full batch expression (blue) is the average of 4 different batches (green, magenta, cyan, yellow), each construct own expressions of the loss function and derivative with different characteristics. (c,d) When sampling randomly between these 4 batches, the function value and corresponding derivative alternates between mini-batch loss function expressions. (e,f) The resulting loss function is discontinuous and non-smooth for function values and derivatives, which is readily interpreted as noise. 23

2.3 (a,c,e,g) Static and (b,d,f,h) dynamic mini-batch sub-sampling (MBSS) loss functions plotted along two random directions using batch sizes $M = |\mathcal{B}| = 150$ and $|\mathcal{B}| \in \{149, 10, 1\}$. Subscripts, n and i, are omitted in plot legends in the interest of compactness. A bias-variance trade-off: Static MBSS produces continuous loss functions, which are biased to the sampling error in $\mathcal{B}_n$; dynamic MBSS represents $\mathcal{L}(\boldsymbol{x})$ on average, but introduces discontinuities by re-sampling $\mathcal{B}_{n,i}$ at every loss evaluation. 27

2.4 A comparison between static and dynamic MBSS as applied in gradient-only line searches [Kafka and Wilke, 2019], showing (a) training and (b) test classification error during training of the MNIST [Lecun et al., 1998] dataset using the NetI [Mahsereci and Hennig, 2017] network architecture. 28

2.5 (a,c,e,g) Function value and (b,d,f,h) directional derivative plots along two orthogonal random directions, $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$, using batch sizes $|\mathcal{B}_{n,i}| \in \{150, 149, 10, 1\}$. Variance for both $\mathcal{L}(\boldsymbol{x})$ and $\tilde{D}_d(\boldsymbol{x})$ increases with decrease in batch size, with $\tilde{D}_d(\boldsymbol{x})$ being less affected than $\mathcal{L}(\boldsymbol{x})$. 30

2.6 Estimated mean and standard deviation plots of (a) $\tilde{L}(\boldsymbol{x})$ and (b) $\tilde{\boldsymbol{g}}(\boldsymbol{x})$. In (a) $\mathcal{L}(\boldsymbol{x})$ is given in red, while in (b) the red plane denotes $d_d(\boldsymbol{x}) = 0$, to visually aid locating sign changes. Viewing these plots from below (c,d) illustrates the spatial spread of potential optima. Instances where $\mathbb{E}[\tilde{L}(\boldsymbol{x})]$ dips below $\mathcal{L}(\boldsymbol{x})$ (green areas) in (c) denote potential local minima. In (d) the green regions denote $\tilde{D}_d(\boldsymbol{x}) < 0$ and red areas $\tilde{D}_d(\boldsymbol{x}) > 0$ respectively. The white shaded areas in (d) are areas of uncertainty, where SNN-GPPs can occur. SNN-GPPs are highly localized, while minima are spread uniformly across the sampled domain. Isolating (d) $\sigma(\tilde{L}(\boldsymbol{x}))$ and (f) $\sigma(\tilde{D}_d(\boldsymbol{x}))$ surfaces, we observe a practical confirmation the discussion in Section 2.2.2. 32

2.7 Diagram depicting the investigated 1-D directions. 33

2.8 1-D plots of the (a) function value and (b) directional derivative along fixed directions of the 2-D domains. We compare two states, using the full-batch loss with $M = 150$, and dynamic MBSS loss with, $|\mathcal{B}_{n,i}| = 10$. We note the number local minima and SNN-GPP in either case according to the respective optimality criteria. Again, the locations of the local minima are uniformly distributed along the sampled domains for all directions. In the case of the directional derivative, directions that contain high curvature, such as $\boldsymbol{d}_1$, have highly localized sign changes. Direction $\boldsymbol{d}_2$ progresses along the area of uncertainty in Figure 2.6 for some of its domain, which makes the location of SNN-GPP more widespread. . . 34

2.9 (a,c) Function value and (b,d) derivatives of activation functions considered in our investigations. These are grouped together into (a,b) saturation and (c,d) sparsity classes respectively. The saturation class makes use of the non-linearities of the activation functions to construct mappings. The sparsity class makes use of the architecture in the network to introduce the non-linearities, where specific "channels" in the network are selectively activated, depending on whether the activation function transmits information or not. 35

2.10 Function values and directional derivatives along fixed search directions using mini-batch size $|\mathcal{B}_{n,i}| = 50$. The increase in batch size reduces the severity of sampling error. This is particularly effective in low curvature directions, causing the locations of SNN-GPPs to be more consistent. However, the localization of local minima is not positively affected by this increase in $|\mathcal{B}_{n,i}|$. 40

3.1 Illustration of the (a) full-batch loss function (blue) and four approximate loss functions computed using four static mini-batches along a descent direction $\boldsymbol{d}_n$. Similarly, (b) indicates the directional derivatives of these four static mini-batch sub-sampled loss functions along $\boldsymbol{d}_n$. 44

3.2 (a) Univariate loss function and (b) derivative function. The non-negative gradient projection point (NN-GPP), x_{nngpp}, (Formulation 3) for the example univariate function coincides with $x_{nngpp} = x_2$, at which the derivative is also zero (Formulation 2) and the function value is a minimum (Formulation 1). NN-GPPs are defined using an optimality formulation that requires all directional derivatives of points in a non-empty ball, $B_\epsilon(x) = \{x \in \mathbb{R} : |x - x_{nngpp}| < \mu,\ 0 < \mu < \infty,\ x \neq x_{nngpp}\}$, around the NN-GPP to be non-negative. 45

3.3 Typical loss function along the steepest descent direction in neural network training, with (a) function value and corresponding (b) directional derivative along the steepest descent direction as a function of step size α. The problem is constructed using the full training data of the Glass1 dataset [Prechelt, 1994] (for details see Section 3.3). The local minimum (Formulation 1) is indicated as a black dot in (a), while the stationary point (Formulation 2) and NN-GPP (Formulation 3) are indicated by a black dot in (b). 46

3.4 Illustration of full-batch and dynamic mini-batch sub-sampled (MBSS) loss function and directional derivative along a unit descent direction, $\boldsymbol{u}$. For every evaluation of $L(\boldsymbol{x})$ and $\boldsymbol{g}(\boldsymbol{x})$ a new mini-batch is sampled. This introduces discontinuities in the loss and directional derivative functions, as every mini-batch, $\mathcal{B}$, has a different sampling error. This variance in sampling error is not present in the full-batch sampled loss function, which is smooth and continuous. Note that $\tilde{\boldsymbol{g}}(\boldsymbol{x})$ is analytically computed using back-propagation for a given mini-batch $\mathcal{B}$, which implies that $\tilde{L}(\boldsymbol{x})$ and $\tilde{\boldsymbol{g}}(\boldsymbol{x})$ are computed for the same mini-batch, $\mathcal{B}$. . . . 48

3.5 (a) Loss function and (b) directional derivative along the steepest descent direction with step size α for different mini-batch sizes, $|\mathcal{B}| \in [10, 30, 50]$, stochastically sub-sampled. The spatial variance in SNN-GPPs along the descent direction is indicated by vertical lines for different batch sizes. (c) The number of local minima and SNN-GPP that manifest for mini-batches sizes between $|\mathcal{B}| = 1$ and $M = 108$ sampled over the domain $0 \leq \alpha \leq 10$ divided into 101 increments. The quality of the solutions captured as the (d) variance in location of local minima and SNN-GPP over the domain $0 \leq \alpha \leq 10$ for a batch size of 10. The full-batch solution is at $\alpha^* = 2.5$. 50
3.6 Comparison between exact line searches that (a) minimize, such as the Golden Section (GS) method, versus (b) identify SNN-GPP by isolating sign changes from negative to positive using Gradient-Only Line Search with Bisection (GOLS-B). . 54
3.7 Schematic diagrams for (a) the function value based inexact line search that is based on Armijo's rule (ARLS) and the (b) gradient-only line search that is inexact (GOLS-I) with the directional derivative slopes at the points of interest highlighted in red. Armijo's rule was developed for smooth functions, while the gradient-only inexact approach was developed for discontinuous functions. 55
3.8 Average training, validation and test dataset losses averaged over (a)-(c) single hidden layer networks, (d)-(f) double hidden layer networks and (g)-(i) both single and double hidden layer networks. 62
3.9 Average (a) training, (b) validation and (c) test dataset losses obtained for different line searches for the HAR dataset [Anguita et al., 2012] using the single hidden layer architecture, an example of a dataset where the function value based Golden-Section method was the best performer. 64
3.10 Average (a) training, (b) validation and (c) test dataset losses obtained for different line searches for the Gene1 dataset [Prechelt, 1994], an example of the Gradient-Only Line Search with Bisection as the best performer. 64
3.11 Average (a) training, (b) validation and (c) test dataset losses obtained for different line searches for the Mushroom1 dataset [Prechelt, 1994], an example of the gradient-only line search that is inexact method as the best performer. 65
3.12 Step size investigations for (a)-(c) the Glass1 dataset [Prechelt, 1994] and (d)-(f) the Cancer1 dataset [Prechelt, 1994] used with the double hidden layer network architecture. Shown are average step sizes, training losses and validation losses obtained for different line search methods. The step sizes are given as a function of iterations, as step sizes are resolved on an iteration basis. The training and test losses are given in the function value domain to remain comparable with Figures 3.8-3.11. This illustrates that GOLS-I has comparable characteristics to GOLS-B in resolving step sizes, but at a fraction of the computational cost. . . . 66
3.13 (a) Training loss and (b) step sizes for training a Variational Autoencoder on MNIST [Lecun et al., 1998] using four variants of GOLS-I. GOLS-I estimates step sizes that are small compared to constant step sizes and suffers worse convergence performance. Consequently we implement GOLS-Max (which is more competitive), GOLS-I with $c_2 = 0$ (no immediate accept condition) and GOLS-Back (always accepts first negative direction derivative) in (c) and (d), with (e) and (f) indicating the norm of descent directions and angles between successive descent directions respectively. 68
3.14 Gradient descent optimization along narrow, "pipe"-like valleys can be decomposed into primary and secondary directions of improvement. Larger step sizes result in significant oscillations in primary directions of improvement, but ultimately lead to faster progression along secondary directions. 70

3.15 (a) Training loss and (b) step sizes for training a Variational Autoencoder on MNIST [Lecun et al., 1998], comparing LS-SGD with different inexact GOLS, to GOLS-I Adagrad, which uses scaled search directions. Convergence performance dramatically improves over LS-SGD and larger step sizes can be estimated by GOLS-I Adagrad. 71
3.16 Static vs. dynamic MBSS: (a,e,i) Training and (b,f,j) test dataset classification errors in terms of function evaluations for mini-batch sample sizes, $|\mathcal{B}| \in \{10, 100, 1000\}$, with (c,g,k) estimated step sizes and (d,h,l) corresponding cumulative step sizes. This problem is adopted from [Mahsereci and Hennig, 2017] and implements a 3-hidden-layer network to classify the CIFAR10 test dataset, while training on Batch1 of the training dataset. This problem is limited in the number of function evaluations, resulting in different numbers of performed iterations, based on the efficiencies of the line searches. 72

4.1 Contour plot of a neural network loss along two random perpendicular directions. Finding an optimum along a search direction: Too small constant steps are expensive, while too large constant steps can be unstable during training. Line searches balance performance and stability. 76
4.2 Full and reduced fidelity loss representations with various definitions used to identify optima. Mini-batch sub-sampling (MBSS) can be conducted using static and dynamic approaches, resulting in smooth and point-wise discontinuous loss approximations respectively. The local minimum, non-negative associated gradient projection point (NN-GPP) and stochastic NN-GPP (SNN-GPP) [Kafka and Wilke, 2019b] definitions apply to full-batch and static MBSS losses. However, only the SNN-GPP is effective in localizing optima in dynamic MBSS loss functions. 77
4.3 Comparing univariate loss functions and directional derivatives along a search direction with (a)(b) full-batch and static MBSS evaluated loss functions, and (c)(d) full-batch and dynamic MBSS loss functions. (e) and (f) show a closer comparison of the directional derivatives of both MBSS modes. 79
4.4 (a) Discontinuous stochastic function with (b) derivatives and (c) an alternative interpretation of (a) that is consistent with the associated derivatives given in (b). The function minimizer of $F(\alpha)$ (grey dot) and sign change from negative to positive along α (red dot) are indicated. 81
4.5 (a) Function values and (b) the directional derivatives of the loss function in dimensions x_8 and x_9 for a single hidden layer neural network applied to the Iris dataset [Fisher, 1936]. Directional derivatives are generated using a fixed search direction $\boldsymbol{d}$, where the only non-zero components are x_8 and x_9, equal to $\frac{1}{\sqrt{2}}$. The directional derivative is then evaluated as $\frac{dF_n}{d\alpha} = \boldsymbol{g}(\boldsymbol{x}_n) \cdot \boldsymbol{d}$, to generate the plots. When using full batches, both the function value and the directional derivatives are smooth and continuous functions. (c) Function values and (d) directional derivatives are discontinuous, when dynamic MBSS with mini-batch size $|\mathcal{B}_{n,i}| = 10$ is implemented. The function value plot's shape is not recognizable in comparison to (a), while directional derivatives still contain features of the original shape. 82

4.6 (a) Function values and (b) directional derivatives along the full-batch steepest descent direction, $\boldsymbol{d}_n = -\nabla\mathcal{L}(\boldsymbol{x})$. The loss function is obtained from the Iris classification problem of Figure 4.5 [Fisher, 1936]. The search direction is sampled by 100 points with sample sizes ranging from $|\mathcal{B}_{n,i}| = 10$ to $|\mathcal{B}_{n,i}| = M = 150$. This is repeated 100 times and the average number of minima and SNN-GPP found at every point is plotted. Minima are spread across the entire domain for most sample sizes in (a). Both minima and SNN-GPPs identify the true optimum, when the full batch is used The spatial variance of SNN-GPP is bounded around the true optimum with increasing spread for decreasing sample size. However, even with the smallest batch size, $|\mathcal{B}_{n,i}| = 10$, SNN-GPPs remain spatially bounded, unlike local minima, which are approximately uniformly spread along the sampled domain. 83
4.7 Schematic diagram of the Gradient-Only Line Search that is Inexact (GOLS-I): Dynamic MBSS results in discontinuous loss functions. $\tilde{F}'(\alpha_{n,0})$ is tested on the initial accept condition, Equation (4.8), if this holds, $\alpha_{n,0}$ is accepted. Otherwise, the directional derivative sign of $\tilde{F}'(\alpha_{n,0})$ determines whether the step size needs to be increased by Equation (4.9) or decreased by Equation (4.10) until an SNN-GPP is isolated. 84
4.8 Log training error, log training loss, log test error and the log of the step sizes as obtained with various batch sizes for the BCWD dataset problem. 91
4.9 Log training error, log training loss, log test error and the log of the step sizes as obtained with various batch sizes for the logistic regression problem for the BCWD dataset. 92
4.10 log Training error, log training loss, log test error and the log of the step sizes as obtained with various batch sizes for the MNIST dataset with the NetI architecture. 93
4.11 Log training error, log training loss, log test error and the log of the step sizes as obtained with various batch sizes for the MNIST dataset with the NetII architecture. 94
4.12 Training error, test error and the log of step sizes as obtained with various batch sizes for the CIFAR10 Dataset, as used with the NetI architecture. The training and test errors are shown on a linear scale to allow comparison with results presented by Mahsereci and Hennig [2017]. 95
4.13 Training error, test error, log of test loss and the log of step sizes as obtained with various batch sizes for the CIFAR10 Dataset, as used with the NetII architecture. The training error is shown on a linear scale to allow comparison to results produced by Mahsereci and Hennig [2017]. 96
4.14 Training loss, step size, training classification error and test classification error for the Cancer dataset problem with a modified NetPI architecture. We extend NetPI to contain 10 hidden layers, thereby making the training problem more non-linear, which prompts the line search algorithms to perform more function evaluations per iteration. 99

5.1 Dynamic mini-batch sub-sampling introduces discontinuities into the loss and gradients of a neural network. However, gradient information, depicted by the directional derivative along a unit direction of two weights, is less affected by discontinuities during resampling. Gradient-only line searches exploit this property to determine step sizes with the aid of the gradient-only optimality condition [Wilke et al., 2013, Snyman and Wilke, 2018, Kafka and Wilke, 2019b, Kafka and Wilke, 2019]. 103
5.2 Not all directions are created equal. Different assumptions and paradigms result in a plethora of optimization strategies with varying search direction trajectories. Step sizes along these directions can be determined using line searches. 106
5.3 Summary of loss surfaces, static and dynamic mini-batch sub-sampling (MBSS) and the performance of minima vs SNN-GPPs. 108

5.4 Comparison between the estimated PDFs of (a) local minima and (b) SNN-GPPs along the steepest descent direction using uniformly sampled mini-batches, for the Iris dataset with mini-batch sizes ranging from $|\mathcal{B}_{n,i}| = 10$ to $M = 76$ (subscripts omitted for compactness). The x-axis indicates the step size α in 100 increments along $\boldsymbol{d} = -\nabla\mathcal{L}(\boldsymbol{x})$ from a fixed starting point. The y-axis shows the log of the probability of encountering a local minimum or SNN-GPP along the search direction. SNN-GPPs are more localized around the true optimum at ≈ 0.076 with in accuracy proportionate to sample size. Local minima are approximately uniformly distributed about the sampled domain for all sample sizes but M. 110

5.5 LS-SGD employed with fixed step sizes and GOLS-I, as applied to (a)-(c) the "Forests" [Johnson et al., 2012], (d)-(f) "Sensor-less Drive" [Paschke et al., 2013] and (g)-(i) "Simulation Failure" Lucas et al. [2013] datasets. GOLS-I successfully adapts step sizes according to the characteristics of the problem. 114

5.6 The "Forests" [Johnson et al., 2012] problem as trained using (a)-(c) LS-NAG, (d)-(f) LS-Adadelta and (g)-(i) LS-Adam with both GOLS-I and fixed step sizes. GOLS-I automatically recovers step sizes close to medium fixed steps for the shown algorithms, while each algorithm has different medium fixed step sizes, each an order of magnitude apart. 115

5.7 The training, test and validation dataset errors for (a)-(c) LS-SGD, (d)-(f) LS-SGD with momentum and (g)-(i) LS-NAG, averaged over all datasets for 10 different starting points per analysis. The algorithms are implemented with different fixed step sizes and the gradient-only based GOLS-I. 117

5.8 The training, test and validation dataset errors for (a)-(c) LS-Adagrad, (d)-(f) LS-Adadelta and (g)-(i) LS-Adam, averaged over all datasets for 10 different starting points per analysis. The algorithms are implemented with different fixed step sizes and the gradient-only based GOLS-I. 118

5.9 Average training, test and validation dataset errors over all datasets and all algorithm in terms of (a)-(c) iterations and (d)-(f) function evaluations. For GOLS-I, the number of function evaluations per iteration is on average 2.5, whereas that of a fixed step size approach is always 1. This is also a proxy for computational cost. Therefore, plotting in terms of function evaluations corrects for this disparity. Though GOLS-I is a factor of 2.5 more expensive than the medium fixed step on average, it remains competitive, while requiring no tuning. 119

5.10 Training, test and validation dataset losses of (a)-(c) GOLS-I SGD, (d)-(f) GOLS-I Adagrad, (g)-(i) GOLS-I Adam and (j)-(l) GOLS-I Adam with $\beta_1 = 0$ for 1-6 of hidden layers. The averages are taken over a subset of datasets with 10 separate training runs per dataset. 120

5.11 The (a) training classification accuracy, (b) step sizes, (c) training classification accuracy in terms of function evaluations and (d) test classification accuracy for various training algorithms as applied to the CIFAR10 dataset with the ResNet18 architecture. The step sizes are determined by GOLS-I, while the explicit reference to "GOLS-I" is omitted in the legends for compactness. explicit The step sizes are different between algorithms, while their performances per iteration are comparable. This indicates that step sizes and not search directions may be the bottleneck for training this problem. However, the cost of each algorithm is not the same, revealing GOLS-I LBFGS to be the worst performer in terms of computational efficiency. 121

5.12 The (a) training loss in terms of iteration, (b) training loss in terms of function evaluations, (c) step sizes for various training algorithms as applied to variational autoencoder [Zuo and Chintala, 2018] training. The step sizes as determined by GOLS-I are again different for every algorithm (with "GOLS-I" omitted in the legends for compactness), while highlighting the effectiveness of Hessian approximations generated by LS-LBFGS. 121

6.1 (a,c) Function value and (b,d) derivatives of activation functions considered in our investigations. These are grouped together into (a,b) saturation and (c,d) sparsity classes respectively. The primary difference between saturation and sparsity classes are the derivatives in the positive input domain. The saturation class is characterized by derivatives that tend towards zero as input tends toward $+\infty$. Conversely, the sparsity class, is characterized by unit derivatives that spread over all of the positive input domain. This gives the sparsity class AFs behaviour characteristics that approximate a "switch", being either "on" or "off" 125
6.2 Contours of the mean squared error loss function along two random perpendicular unit directions, $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$, computed for a single hidden layer feedforward neural network using different activation functions. The hidden layer consists of 10 hidden nodes and the architecture is applied to the classic Iris dataset [Fisher, 1936]. 126
6.3 Univariate functions and directional derivatives of (a) $\mathcal{F}_n$ and (b) $\mathcal{F}'_n$, using full-batch sampling, and (c) $\tilde{F}_n$ and (d) $\tilde{F}'_n$, using dynamic mini-batch sub-sampling with a selection of activation functions. 128
6.4 Training, validation and test losses with corresponding log of step sizes for the foundational problems with various networks architectures, including (a) investigation 1: single hidden layer (HL) networks with $\frac{D}{2}$ hidden nodes, (b) investigation 2: single HL networks with $2D$ nodes and (c) investigation 3: networks with six HLs of $2D$ nodes using GOLS-I Adagrad. For (a) to (c), GOLS-I was used to determine step sizes, while (d) investigation 4 implements LS-SGD with fixed step sizes $\alpha_{n,I_n} = 0.05$ for a single HL network with $2D$ nodes. 133
6.5 Training loss, test loss and the log of step sizes for a subset of datasets (DS) from the foundational problems dataset pool, trained using GOLS-B SGD (a,b) with mini-batch size $|\mathcal{B}_{n,i}| = 64$ for investigation 5 and (c,d) using full-batches, M in investigation 6. 135
6.6 Training and test classification errors with corresponding log of step sizes as obtained for the MNIST Dataset with the NetII architecture for different activation functions. Step sizes for LS-SGD are determined by (a)-(c) GOLS-I for the standard NetII architecture with all considered AFs, (d)-(f) a range of fixed step sizes for the standard NetII architecture with only ReLU, and (g)-(i) determined by GOLS-I for NetII with batch normalization and all considered AFs. 137
6.7 Schematic of the distribution of incoming information to a ReLU activation function (a) at initialization, (b) without batch normalization (BN) after spurious updates due to step sizes that are too large, and (c) after implementing batch normalization. Batch normalization centres the incoming information distribution around 0 and scales it, ensuring that the ReLU activation function remains active with a reasonable probability. Subsequently, $\tilde{\boldsymbol{g}}(\boldsymbol{x})$ is more likely to remain dense than with the standard feedforward architecture. 139
6.8 (a) Log training error, (b) log test error and (c) the log of step sizes for the CIFAR10 Dataset with the ResNet18 architecture trained using GOLS-I SGD. The standard ResNet architecture includes batch-norm layers, which are omitted in (d)-(f) in order to highlight the effect of skip connections on training with sparsity enforcing ReLU activation functions. 140
6.9 Illustration of the structural difference between (a) standard feedforward nodes, and (b) skip-connected nodes. 142

A.1 The Tanh AF: A steeper AF derivative around 0 means more curvature in the loss gradients, which manifest as steeper directional derivatives. This aids SNN-GPP localization in the active region (origin of (h)), but not at saturation (outer ranges of (h)). 148

A.2 The Softsign AF: The less aggressive taper-off in the AF derivative reduces the chance for spurious SNN-GPPs in the saturation regions. In the centre domain, SNN-GPPs are highly localized around the full-batch, true optima. 149
A.3 The ReLU AF: Pushing ReLU activations far into their active domain results in convex behaviour of the MSE loss function on a large scale. However, directional derivative and true optima plots indicate that more detail is contained in the basin. 150
A.4 The leaky ReLU AF: Since the magnitude of the "leaky" gradient is relatively small, its contribution is not apparent at this length scale. Therefore, the plots look very similar to those of ReLU. SNN-GPPs are concentrated around the centre, where the magnitude of the directional derivatives is small. 151
A.5 The ELU AF: The shapes of the convex element of the loss function are similar to those of the other ReLUs. However, the structure in the basin seems to be different. This is confirmed by the number of true optima. The narrow spatial grouping of SNN-GPPs is the contribution of the continuous AF derivative. . . . 152
A.6 The Sigmoid AF close-up: Shapes are smooth and have little curvature. SNN-GPPs have a smaller spatial range in directions where the curvature is larger, while being more spread out in low curvature directions. 153
A.7 The Tanh AF close-up: The higher curvature of Tanh helps localize SNN-GPPs. Though this example captures lower variance directions in the function value (see (g), $\boldsymbol{d}_2$), this does not alleviate the problem of uniformly spread spurious local minima. 154
A.8 The Softsign AF close-up: The small domain investigation confirms that the less aggressive taper-off of the AF derivative contributes to localizing SNN-GPPs around the true optimum, while avoiding spurious instances at saturation. Local minima remain uniformly distributed. 155
A.9 The ReLU AF close-up: Notable features are the stark changes in the directional derivatives. These correspond to the "activation" and "deactivation" of various nodes in the network. A unique feature to the ReLU activation is the presence of flat planes where the directional derivative is 0. These areas denote weight spaces where no information passes through the network, for all mini-batches. 156
A.10 The leaky ReLU AF close-up: Flat planes with constant directional derivative values are also present here. Although in this case they have a non-zero numerical value, they do not contribute significantly to localizing SNN-GPPs or reducing the number of true optima for the ReLU class. 157
A.11 The ELU AF close-up: The smooth exponential derivative in the negative domain of the AF results in a greater magnitude in negative directional derivatives in the loss ((d) and (h)) compared to the remainder of the sparsity class. This in turn helps distance the directional derivatives from 0, benefiting localization of SNN-GPPs compared to leaky ReLU. 158
A.12 Detailed comparison of directional derivative plots between ReLU and leaky ReLU activations when hidden units become "inactive". As expected, ReLU units switch "off" entirely, containing no gradient information, while the leaky ReLU results in non-zero directional derivatives. 159

List of Tables

2.1 Summary of observations from analyses conducted by traversing random directions over a scalar range of [-20,20] to give an impression of "global" loss function characteristics. The corresponding Figures A.1-A.5 are included in Appendix A.1.1. 37

2.2 Summary of observations from analyses conducted by traversing a scalar range of [-2,2] to give an impression of "local" loss function characteristics. Here the search directions are $\boldsymbol{d}_1 = -\nabla\mathcal{L}(\boldsymbol{x})$ and $\boldsymbol{d}_2$ is a random perpendicular direction to $\boldsymbol{d}_1$. The corresponding Figures A.6-A.11 are shown in Appendix A.1.2. 38

2.3 Number of local minima and SNN-GPPs counted in different domain sizes as a function of mini-batch size, $|\mathcal{B}_{n,i}|$. 40

3.1 Properties of the considered datasets for the investigation. 60

3.2 Properties of the neural network architecture for each dataset. 60

3.3 Average number of function evaluations and information calls per iteration for the various line searches. 67

3.4 Mean angles between consecutive search directions $\boldsymbol{d}_{n-1}$ and $\boldsymbol{d}_n$ for various gradient-only line search methods with LS-SGD. 69

4.1 Relevant parameters related to the datasets used in numerical experiments. . . . 89

4.2 Parameters and settings governing the implemented network architectures and their training. 90

4.3 Various metrics of function evaluations (Fe.) performed per iteration (It.) during training for PrLS and GOLS-I. Function evaluations performed during an iteration indication of the "effort" exerted by the line search to determine the step size. The maximum number of Fe./It. in PrLS is fixed, which is not the case for GOLS-I. The absolute maximum number of Fe./It. for GOLS-I is 28, which occurs in the first iteration due to a conservative initial guess of $\alpha_{0,0} = 1 \cdot 10^{-8}$. In subsequent training, the maximum number is 11. 100

5.1 Learning rate ranges as used for different optimization algorithms. 113

5.2 Properties of the datasets considered for the investigation with the corresponding neural network architecture. 113

5.3 Algorithmic comparison in terms of cost. A constant number of iteration was used to simplify the comparison of training performance. However, not all algorithms use the same number of function evaluations per iteration with GOLS-I coupled to different algorithms. Each function evaluation consists of calling only the gradient, $\tilde{\boldsymbol{g}}(\boldsymbol{x})$. 122

5.4 Maximum achieved test classification accuracy after 40,000 training iterations on the CIFAR10 dataset with various training algorithms in combination with GOLS-I. 122

6.1 Properties of the datasets considered for conducted investigations. 131

6.2 Parameters and settings governing the implemented network architectures and their training for various investigations (Inv.). 132

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
GOLS	Gradient-only line searches
GOLS-I	Gradient-Only Line Search that is Inexact
GOLS-B	Gradient-Only Line Search with Bisection
GOLS-Max	Gradient-Only Line Search that Maximizes step size
GOLS-Back	Gradient-Only Line Search with Backtracking
ARLS	Armijo's Rule Line Search
GS	Golden Section (Line Search)
PrLS	Probabilistic Line Search
SGD	Stochastic Gradient Descent algorithm
SGDM	Stochastic Gradient Descent with Momentum algorithm
NAG	Nesterov Accelerated Gradient Descent algorithm
LBFGS	Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm
LS-SGD	Line Search Stochastic Gradient Descent algorithm
LS-SGDM	Line Search Stochastic Gradient Descent with Momentum algorithm
LS-NAG	Line Search Nesterov Accelerated Gradient Descent algorithm
LS-Adagrad	Line Search Adagrad algorithm
LS-Adadelta	Line Search Adadelta algorithm
LS-Adam	Line Search Adam algorithm
LS-LBFGS	Line Search LBFGS algorithm
AF	Activation function
IAC	Initial-accept condition

List of Symbols

$\boldsymbol{x}$	Vector of neural network weights, where $\boldsymbol{x} \in \mathbb{R}^p$
$\boldsymbol{t}_b$	Training dataset observation pair (input and output) for observation b
M	Number of samples in training set $T = \{\boldsymbol{t}_1, \dots, \boldsymbol{t}_M\}$
$\ell(\boldsymbol{x};\ \boldsymbol{t}_b)$	Elemental loss for a single training data sample
$\mathcal{L}(\boldsymbol{x})$	Full-batch loss function evaluated over all M training samples
$\nabla\mathcal{L}(\boldsymbol{x})$	Gradient of $\mathcal{L}(\boldsymbol{x})$
$\cdot^*$	An optimum of a smooth function
n	Iteration number counter, relating to an optimization/training algorithm
i	Function evaluation counter within an iteration (also linked to increments)
$\mathcal{B}$	Mini-batch as a subset of training data, T
$\lvert\mathcal{B}\rvert$	The number of samples in mini-batch $\mathcal{B}$, where $\lvert\mathcal{B}\rvert < M$
$\boldsymbol{d}_n$	The search direction of an algorithm at iteration n
$\mathcal{B}_n$	Static MBSS batch, re-sampled at every iteration n of an algorithm and constant over a line search
$\mathcal{B}_{n,i}$	Dynamic MBSS batch, re-sampled at function evaluation i within an iteration n of an algorithm
B_ϵ	Ball of dimensionality p within which all SNN-GPPs of a loss neighbourhood are contained
$L(\boldsymbol{x})$	Approximate loss function evaluated using $\mathcal{B}$
$\boldsymbol{g}(\boldsymbol{x})$	Approximate gradient evaluated using $\mathcal{B}$
$\bar{L}(\boldsymbol{x})$	Static MBSS loss evaluated using $\mathcal{B}_n$
$\bar{\boldsymbol{g}}(\boldsymbol{x})$	Static MBSS gradient evaluated using $\mathcal{B}_n$
$\tilde{L}(\boldsymbol{x})$	Dynamic MBSS loss evaluated using $\mathcal{B}_{n,i}$
$\tilde{\boldsymbol{g}}(\boldsymbol{x})$	Dynamic MBSS gradient evaluated using $\mathcal{B}_{n,i}$
F	Univariate loss function
F'	Derivative of F
α	Step size
$\mathcal{F}_n(\alpha)$	Line search evaluated along full-batch loss, $\mathcal{L}(\boldsymbol{x})$
$\mathcal{F}'_n(\alpha)$	Directional derivative of line search $\mathcal{F}_n(\alpha)$
$\bar{F}_n(\alpha)$	Line search evaluated along static mini-batch sub-sampled loss, $\bar{L}(\boldsymbol{x})$
$\bar{F}'_n(\alpha)$	Directional derivative of line search $\bar{F}_n(\alpha)$
$\tilde{F}_n(\alpha)$	Line search evaluated along dynamic mini-batch sub-sampled loss, $\tilde{L}(\boldsymbol{x})$
$\tilde{F}'_n(\alpha)$	Directional derivative of line search $\tilde{F}_n(\alpha)$
I_n	Number of function evaluations for line search at iteration n
α_{n,I_n}	Step size at iteration n determined using I_n function evaluations
$\boldsymbol{u}$	A unit vector
D	Number of input dimensions (features) of a dataset
E	Number of output dimensions (classes) of a dataset
$\boldsymbol{t}_b^i$	Input data vector of observation $\boldsymbol{t}_b$
$\boldsymbol{t}_b^o$	Output data vector of observation $\boldsymbol{t}_b$
$\hat{\boldsymbol{t}}^o$	Output domain vector estimation from a model (neural network)

Chapter 1

Overview

Step sizes in neural network training are largely set using predetermined rules such as fixed learning rates or learning rate schedules. These require user input or expensive global optimization strategies to resolve their functional form and associated hyperparameters. Typically, these learning rate parameters are among the most sensitive hyperparameters in neural network training. Attempts to resolve learning rates automatically using line searches have been hindered, as minimization line searches require loss functions to be continuous and smooth. However, due to growing data demands, memory-limited computational resources such as graphical processing units (GPUs), and the dynamics of on-line learning, make mini-batch sub-sampling in neural network training unavoidable. The omission of training data to produce a given mini-batch introduces a sampling error into the loss function. Although different mini-batches represent the full-batch loss function on average, individual instances might vary significantly from the mean, or full-batch loss function [Tong and Liu, 2005].

In this work we specifically distinguish between static and dynamic mini-batch sub-sampled (MBSS) loss functions. Generally, these two sub-sampling approaches are not explicitly disambiguated between researchers, as they are synonymous in the case where predetermined learning rate schedules are applied. However, as the interest in conducting line searches for neural network training increases, a lack of distinction between sub-sampling approaches *within* a line search can lead to confusion. In static MBSS loss evaluations, mini-batches are fixed for a minimum duration of a line search during training. This presents a line search algorithm with smooth loss functions that have no variance, but distinct bias due to sampling error [Friedlander and Schmidt, 2011, Bollapragada et al., 2017, Kungurtsev and Pevny, 2018]. Conversely, in dynamic MBSS losses, new mini-batches are sampled at every loss evaluation within a line search. This trades a fixed sampling error bias for variance as the sampling error is continuously changing for every mini-batch [Mahsereci and Hennig, 2017, Wills and Schön, 2018, 2019]. Therefore, this distinction is important, as the method of sub-sampling determines how the sampling error manifests within a loss function, namely, either as bias or variance. Commonly, the variance produced by dynamic MBSS is referred to as noise [Simsekli et al., 2019], implying that the stochastic loss component has no structure. After further consideration of both static and dynamic MBSS, we argue that dynamic MBSS with a fixed training dataset results in point-wise discontinuous loss functions that have a deterministically limited number of sampling errors. The range of sampling errors are linked to the number of possible mini-batch combinations in the training dataset. Therefore, as the training dataset and the mini-batch sizes increase, so do the possible combinations of mini-batches to produce different sampling errors. Bottou [2010] argues, that the act of constantly changing the mini-batch during approximate optimization benefits training, as it exposes an algorithm to a larger amount of training data. However, dynamic MBSS renders minimization line searches ineffective, since critical points may not exist and function minimizers find spurious, discontinuity-induced local minima.

Since optimization within smooth and continuous loss functions is a well developed field [Arora, 2011], we decide to shift our attention to optimization in dynamic MBSS loss functions instead, as it dominates training problems encountered in machine learning. Training neural networks in particular is a rapidly developing field with wide scope for application. However, applying matured methods from fields such as mathematical programming to aid neural network training requires, at a minimum, the ability to reliably identify candidate solutions close to optima of full-batch loss functions. As mentioned, the discontinuous nature of dynamic MBSS losses restricts the ability of traditional optimality criteria such as local minima and critical points to find such candidate solutions.

Instead, we suggest recasting the optimization problem in dynamic MBSS losses to find Stochastic Non-Negative Associated Gradient Projection Points (SNN-GPPs). We demonstrate that the SNN-GPP optimality criterion is less susceptible to dynamic MBSS induced discontinuities than critical points or minimizers. Along a univariate descent direction, as used in line searches, an SNN-GPP is identified by a sign change from negative to positive in the directional derivative. We demonstrate that SNN-GPPs can be used to great effect in Gradient-Only Line Searches (GOLS) to determine step sizes in the dynamic MBSS loss functions of neural network training. This allows learning rate schedules to be resolved adaptively during training, effectively eliminating the need for expensive parameter tuning. The ability of GOLS to determine step sizes is independent of the algorithm used. Different algorithms can produce a variety of distinct search directions, while GOLS subsequently determine step sizes to estimate the location of full-batch optima along the given search directions.

The purpose of this work is to progressively develop GOLS for dynamic MBSS losses in neural network training, starting from building a visual intuition of the SNN-GPP optimality criterion, to comparing GOLS methods to existing benchmarks such as minimization and probabilistic line searches; and finally, exploring the characteristics of GOLS as applied in practice for different training algorithms and activation functions. This work contributes to machine learning optimization, not only by eliminating the need for learning rate schedules, but more importantly by strengthening the ties between mathematical programming and machine learning. The introduction of effective line searches in dynamic MBSS losses allows more training methods from classical optimization theory to be applied to neural network training. This promotes popular classic optimization approaches such as Quasi-Newton methods [Arora, 2011] to be explored more seriously in the context of dynamic MBSS neural network losses. Additionally, GOLS can be used as a research tool to further explore the nature of neural network loss functions. This study is divided into five additional self contained chapters, each exploring different aspects related to gradient-only line searches. Each of these chapters are based on bodies of work that have been submitted and/or accepted for publication in various journals.

In Chapter 2[Kafka and Wilke, 2019a] we conduct a visual investigation comparing local minimum and SNN-GPP optimality criteria in the loss functions of a foundational neural network training problem. We consider a variety of popular activation functions in the selected feedforward network architecture. We show that SNN-GPPs better approximate the location of full-batch optima, particularly when using smooth activation functions with high curvature characteristics. This suggests that it is feasible to construct line searches that locate SNN-GPPs, which can contribute significantly to automating neural network training.

Chapter 3[Kafka and Wilke, 2019b] develops a robust approach to adaptively resolve learning rates in dynamic MBSS loss functions. Over a number of investigative studies we demonstrate that gradient-only line searches (GOLS) are able to adaptively resolve learning rate schedules. We show that GOLS outperform minimization line searches over a wide range of foundational training problems with different network architectures. Additionally, we demonstrate that poor search directions may artificially benefit from overstepping optima along a descent direction. Conversely, improving search directions can lead to comparable performance to overstepping optima. After establishing GOLS as a reliable line search approach, and using the property that SNN-GPPs generalize to smooth functions, we can for the first time directly compare dynamic and static MBSS within line searches for neural network training.

Subsequently, in Chapter 4[Kafka and Wilke, 2019] we benchmark the Gradient-Only Line search that is Inexact (GOLS-I) (as developed in Chapter 3) against Probabilistic Line Searches (PrLS) [Mahsereci and Hennig, 2017]. PrLS uses statistical surrogates to determine step sizes in dynamic MBSS losses, which comes with a significant amount of overhead. We show that GOLS-I is a competitive strategy to reliably resolve step sizes, adding high value in terms of performance, while being comparatively easy to implement. We show that PrLS is effective in determining step sizes when mini-batches are very small (≈ 10), while GOLS-I becomes increasingly competitive and outperforms PrLS as the mini-batch size grows (≤ 50) for the investigated training problems.

Having demonstrated the capabilities of GOLS-I as a line search method in dynamic MBSS loses, we study its ability to automatically determine the learning rate schedule for a selection of popular neural network training algorithms in Chapter 5. These include NAG, Adagrad, Adadelta, Adam and LBFGS, which are applied to a number of shallow and deep, as well as a convolutional neural network architecture, trained on different datasets with various loss functions. We find that overall, GOLS-I's learning rate schedules are competitive with manually tuned learning rates over seven optimization algorithms, three types of neural network architecture, 23 datasets and two types of loss function. This showcases GOLS-I's ability to automatically adapt step sizes to a variety of training algorithms and problems. However, we note that algorithms which include dominant momentum characteristics are not well suited to be used with GOLS-I. In algorithms such as Adam, where this momentum behaviour can be eliminated by changing algorithm parameters, the competitive performance of GOLS-I can be immediately restored.

Lastly, we investigate the relationship between activation functions with different neural network architectures and the training performance of GOLS in Chapter 6. Activation functions are a significant component of neural network training problems. Their smoothness and continuity characteristics directly effect loss function characteristics and thus the gradient information used by GOLS. We find that GOLS are robust over a large range of activation functions, but caution is advised when implementing Rectified Linear Unit (ReLU) activation functions in standard feedforward architectures. The zero-derivative in ReLU's negative input domain can lead the gradient-vector to become sparse, which effectively stalls training. We demonstrate, that implementing architectural features such as batch normalization and skip connections can alleviate the difficulties experienced when training ReLU using GOLS. Additionally, these features also benefit the remaining activation functions considered, by accelerating training overall.

This study is concluded in Chapter 7 and recommendations to stimulate future study are offered. An overview of the major contributions of this work is given as follows: Firstly, we explicitly define static and dynamic mini-batch sub-sampling in the context of loss landscapes in Chapter 2 and line searches in Chapter 3. The separation of these sub-sampling approaches allows us to focus explicitly on optimization of stochastic training problems using dynamic MBSS from Chapter 3 onwards. This directed consideration prompts us to extend the gradient-only optimality criterion [Wilke et al., 2013, Snyman and Wilke, 2018] to the *Stochastic Non-Negative Associated Gradient Projection Point*. Subsequently, we develop four formulations of GOLS in Chapter 3 to find SNN-GPPs with the purpose of determining step sizes in dynamic MBSS losses. We benchmark GOLS against minimization line searches and probabilistic line searches [Mahsereci and Hennig, 2017] in Chapters 3 and 4 respectively. Finally, we explore the performance characteristics of GOLS in the context of different training algorithms in Chapter 5 and neural network architectures with a variety of activation functions in Chapter 6.

In the interest of transparency and reproducibility, we make our code available at `https://github.com/gorglab/GOLS`. The repositories include user-friendly versions of the source code used to conduct the investigations in this study. Examples are available in both Matlab [Mathworks, 2015] and PyTorch 1.0 [pytorch.org, 2019], which provide a range of training problems that can be solved using different combinations of GOLS and training algorithms. This collection of examples demonstrates the ability of GOLS to determine step sizes for a sizeable assortment of training problems, while also providing a platform with which to explore further characteristics of GOLS.

Chapter 2

Traversing the noise of dynamic mini-batch sub-sampled loss functions: A visual guide

Mini-batch sub-sampling in neural network training is unavoidable, due to growing data demands, memory-limited computational resources such as graphical processing units (GPUs), and the dynamics of on-line learning. In this chapter we specifically distinguish between static mini-batch sub-sampled loss functions, where mini-batches are intermittently fixed during training, resulting in smooth but biased loss functions; and the dynamic sub-sampling equivalent, where new mini-batches are sampled at every loss evaluation, trading bias for variance in sampling induced discontinuities. These render automated optimization strategies such as minimization line searches ineffective, since critical points may not exist and function minimizers find spurious, discontinuity induced minima.

In this chapter, we suggest recasting the optimization problem to find stochastic non-negative associated gradient projection points (SNN-GPPs). We demonstrate that the SNN-GPP optimality criterion is less susceptible to sub-sampling induced discontinuities than critical points or minimizers. We conduct a visual investigation, comparing local minimum and SNN-GPP optimality criteria in the loss functions of a simple neural network training problem for a variety of popular activation functions. Since SNN-GPPs better approximate the location of true optima, particularly when using smooth activation functions with high curvature characteristics, we postulate that line searches locating SNN-GPPs can contribute significantly to automating neural network training.

2.1 Introduction: The stochastic neural network optimization problem

The training of neural networks centres around minimizing loss functions that commonly take the form of

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

where $T = \{\boldsymbol{t}_1, \ldots, \boldsymbol{t}_M\}$ is the training dataset of M samples, and the model parameters are given by vector $\boldsymbol{x} \in \mathbb{R}^p$. The loss quantifying the adequacy of parameters $\boldsymbol{x}$ in terms of training data T is given by $\mathcal{L}(\boldsymbol{x})$. The gradient of the loss function with regards to parameters $\boldsymbol{x}$ is given by

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

which is computed efficiently using backpropagation [Werbos, 1994]. The loss surface of $\mathcal{L}(\boldsymbol{x})$ is continuous, while the continuity of $\nabla\mathcal{L}(\boldsymbol{x})$ depends on the continuity and smoothness of the activation function (AF) used.

A local minimum of a function $f(\boldsymbol{x})$ is defined as follows:

Definition 2.1.1. *Local Minimum: Let $\boldsymbol{x}^*$ be a local minimum of $f(\boldsymbol{x})$, such that*

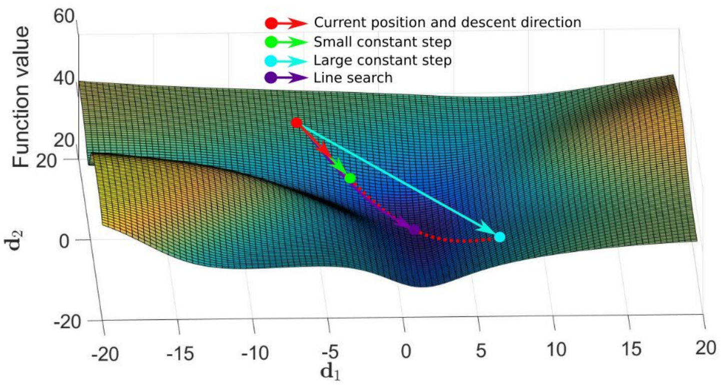
$$\Delta f(\boldsymbol{x}) = f(\boldsymbol{x}) - f(\boldsymbol{x}^*) \geq 0, \tag{2.3}$$

for any point $\boldsymbol{x}$ in the neighbourhood of $\boldsymbol{x}^*$ [Arora, 2011].

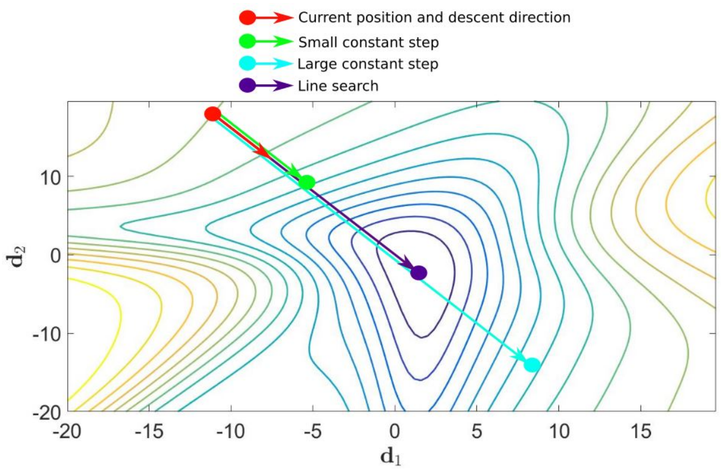
In order to find minima, consider the update step of the popular Gradient Descent (GD) algorithm, given as

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \alpha\nabla\mathcal{L}(\boldsymbol{x}_n), \tag{2.4}$$

at a given iteration, n. Here the update is a function of a search direction $\boldsymbol{d}_n = -\nabla\mathcal{L}(\boldsymbol{x}_n)$, the gradient of the loss function at the current location, and an undetermined step size, scalar α. A visual representation of selecting the step size along a gradient descent search direction is depicted in Figure 2.1. If α is too small, insufficient progress is made along the descent direction, causing slow neural network training. If α is too large, the minimum along the search direction is overshot, and training can become unstable. Line searches are common methods employed in mathematical programming [Nocedal and Wright, 1999, Wächter and Biegler, 2005, Nie, 2006, Arora, 2011] to resolve the step size, α, up to a desired accuracy, balancing training performance and stability. Importantly, line searches perform best when the full dataset is available to evaluate the loss function.



(a) GD update in a neural network loss function.



(b) Contour plot of an GD update step.

Figure 2.1: Potential gradient descent (GD) updates in a neural network loss function for the Iris [Fisher, 1936] classification problem.

However, nowadays machine learning (ML) training is rarely conducted using full batches [Krizhevsky et al., 2012, Csiba and Richtárik, 2018]. Growing data demands, memory limited efficient computational resources, such as graphical processing units (GPUs), the dynamic world of on-line learning [Mahsereci and Hennig, 2017] and improved convergence characteristics [Saxe et al., 2013, Dauphin et al., 2014, Choromanska et al., 2015], has cemented mini-batch sub-sampling as a *d*e facto standard in ML training. In particular, the use of a smaller number of samples, i.e. $\mathcal{B} \subset \{1, \ldots, M\}$ with $|\mathcal{B}| \ll M$, have become common practice. The approximated loss and gradient functions are given by:

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

and

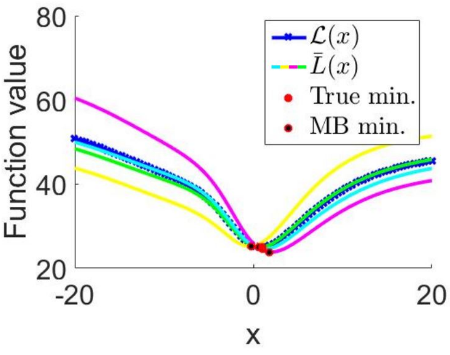
$$\boldsymbol{g}(\boldsymbol{x}) = \frac{1}{|\mathcal{B}|} \sum_{b \in \mathcal{B}} \nabla \ell(\boldsymbol{x};\ \boldsymbol{t}_b). \tag{2.6}$$

For most sampling approaches, these approximations have expectation $\mathbb{E}[L(\boldsymbol{x})] = \mathcal{L}(\boldsymbol{x})$ and $\mathbb{E}[\boldsymbol{g}(\boldsymbol{x})] = \nabla \mathcal{L}(\boldsymbol{x})$ [Tong and Liu, 2005]. However, significant variations from the mean can be observed between expressions of individual batches, $\mathcal{B}$, which we call the *sampling error.* This study distinguishes between two mini-batch sub-sampling (MBSS) approaches used to compute a loss function, namely static and dynamic MBSS. These two MBSS approaches significantly affect the characteristics of the various computed loss functions. This chapter aims to highlight and demonstrate these implications, in addition to developing an intuition for interpreting the available information within the context of line search approaches to resolve learning rates.

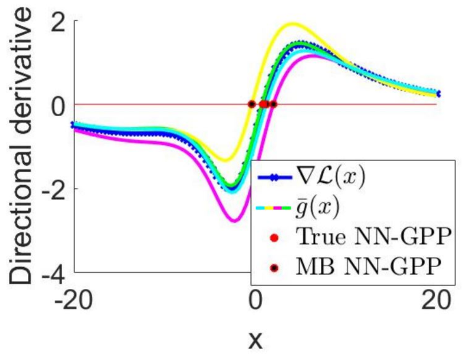
Before we explore static and dynamic MBSS, it is important to explore adaptive sub-sampling methods, that primarily aims to resolve batches or batch sizes with desired characteristics. The aim might be to select a sub-sample such that $\boldsymbol{g}(\boldsymbol{x}) \approx \nabla \mathcal{L}(\boldsymbol{x})$, or to ensure that descent directions computed by $\boldsymbol{d}_n = -\boldsymbol{g}(\boldsymbol{x})$ are indeed mostly descent directions [Friedlander and Schmidt, 2011, Bollapragada et al., 2017]. A carefully selected mini-batch is usually kept constant over a few iterations along a few search directions [Martens, 2010, Friedlander and Schmidt, 2011, Byrd et al., 2011, 2012, Bollapragada et al., 2017, Kungurtsev and Pevny, 2018, Paquette and Scheinberg, 2018, Bergou et al., 2018, Mutschler and Zell, 2019], which can then also be used to conduct line searches to resolve learning rates. We call this or any other approach that keeps a mini-batch fixed along a search direction, $\boldsymbol{d}_n$, static MBSS. In this study, we denote loss and gradient approximations computed using static MBSS by $\bar{L}(\boldsymbol{x})$ and $\bar{\boldsymbol{g}}(\boldsymbol{x})$, respectively. Mini-batches sampled for static MBSS are denoted as $\mathcal{B}_n$.

A 1-D loss function representation is illustrated in Figures 2.2(a) and (b). The training data is split into 4 equally sized, static mini-batches (green, magenta, cyan, yellow), each resulting in a continuous and smooth loss expression, $\bar{L}(\boldsymbol{x})$, with own minimizer, $\boldsymbol{x}^{*\mathcal{B}_n}$, and associated local minimum. The blue curve denotes the true or full-batch loss, $\mathcal{L}(\boldsymbol{x})$. Evidently, the mini-batch minimizers, $\boldsymbol{x}^{*\mathcal{B}_n}$ denoted by the prefix "MB", are not equal to, but occur in a range around the true or full-batch minimizer, $\boldsymbol{x}^{*M}$ denoted by the prefix "True".

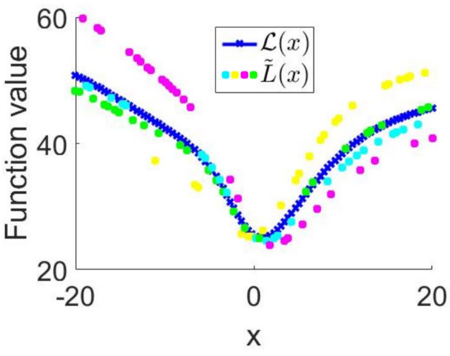
Suppose now that a new loss function is constructed by an oracle [Agarwal et al., 2012], that randomly selects one of the four batches for every increment, i, that the loss function or derivative is evaluated along x as shown in Figures 2.2(c) and (d). We call this dynamic MBSS and differentiate loss function and gradient evaluations using this scheme by $\tilde{L}(\boldsymbol{x})$ and $\tilde{\boldsymbol{g}}(\boldsymbol{x})$, respectively. Mini-batches sampled using dynamic MBSS are denoted $\mathbf{B}_{n,i}$, as re-sampling occurs at every increment, i, within iteration, n, of a potential optimization algorithm. This results in a discontinuous loss function as the oracle randomly selects a loss function sampled by one of the four mini-batches. We show this explicitly in Figure 2.2(c) and (d), by retaining the colours of the contributing mini-batches towards the dynamic MBSS loss function. Here, the contribution of each batch is evident, whereas, plotting the computed dynamic MBSS loss function using a single colour hinders identification of the contribution of a single batch, as shown in Figures 2.2(e) and (f). Note, that there are a finite number of combinations in which



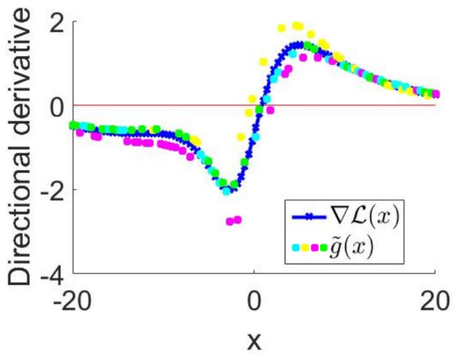
(a) $\mathcal{L}(x)$ and 4 $\bar{L}(x)$ curves



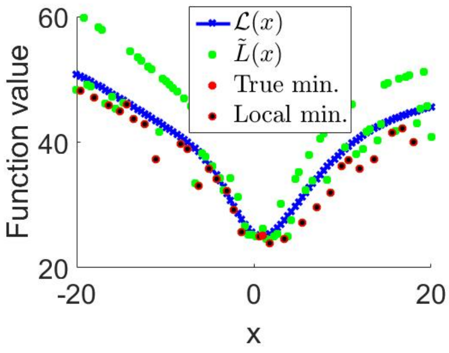
(b) $\nabla\mathcal{L}(x)$ and 4 $\bar{g}(x)$ curves



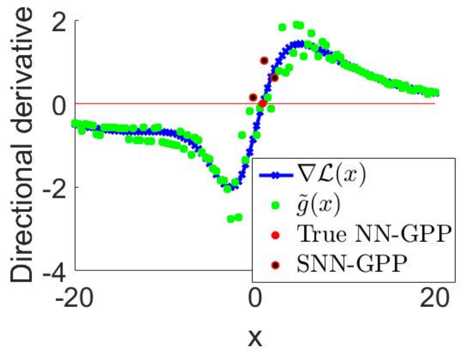
(c) $\tilde{L}(x)$, with selected batches coloured



(d) $\tilde{g}(x)$, with selected batches coloured



(e) $\tilde{L}(x)$ as a single function



(f) $\tilde{g}(x)$ as a single function

Figure 2.2: Conceptual depiction of static and dynamic mini-batch sub-sampled loss functions. (a,b) The full batch expression (blue) is the average of 4 different batches (green, magenta, cyan, yellow), each construct own expressions of the loss function and derivative with different characteristics. (c,d) When sampling randomly between these 4 batches, the function value and corresponding derivative alternates between mini-batch loss function expressions. (e,f) The resulting loss function is discontinuous and non-smooth for function values and derivatives, which is readily interpreted as noise.

the discontinuities can occur for a dataset of fixed size. The number of combinations, in Figures 2.2(e) and (f), were limited to four, but sampling conducted with replacement, there are $K = \binom{M}{|\mathcal{B}_{n,i}|}$ combinations in which a mini-batch $\mathcal{B}_{n,i}$ can be assembled from the training set. Consequently, the number of combinations explode for large M and small batch sizes, which results in these discontinuities being interpreted readily as stochastic noise [Simsekli et al., 2019]. Not surprisingly, these discontinuities hinder attempts to directly minimize dynamic MBSS loss functions using line searches. This is due to sampling induced discontinuities that manifest as local minimizers. Concurrently, critical points typically do not exist at individual samples $\tilde{L}(\boldsymbol{x}^*)$, although they might be present in expectation. This renders ineffective the predominant mathematical programming paradigm for defining optimal as the minimum loss function.

In spite of this, Bottou [2010] argues that approximate optimization methods, which implement dynamic MBSS, can benefit from being exposed to more information. In a recent study, the probabilistic line search developed by Mahsereci and Hennig [2017] estimates step sizes for dynamic MBSS loss functions. Benefits of changing the mini-batch sub-sample at every evaluation include overcoming weaker local basins of attraction [Kleinberg et al., 2018, Simsekli et al., 2019], in addition to acting as an effective regularizer to obtain solutions with better generalization properties [Masters and Luschi, 2018]. This motivated the use of adaptive and dynamic (recursive) sampling [Kashyap et al., 1970, Csiba and Richtárik, 2018], as alluded to earlier, to reduce the impact of these discontinuities. Alternatively, sub-gradient methods [Shor, 1985b], originally developed for continuous non-smooth loss functions, utilizes fixed learning rates or fixed learning rate schedules. Considering a simple modification [Wilke, 2011], they are also able to optimize discontinuous loss functions, as has been done for years in ML training, albeit often unwittingly. In fact, Wilke [2011] demonstrated that sub-gradient methods do not minimize discontinuous loss functions but rather resolve non-negative associated gradient projection points (NN-GPPs). NN-GPP were specifically designed to define optimality for discontinuous functions [Wilke et al., 2013, Snyman and Wilke, 2018], and is given by:

Definition 2.1.2. *NN-GPP: A non-negative associated gradient projection point (NN-GPP) is defined as any point, $\boldsymbol{x}_{nngpp}$, for which there exists $\epsilon_{max} > 0$ such that*

$$\boldsymbol{u} \cdot \nabla f(\boldsymbol{x}_{nngpp} + \epsilon \boldsymbol{u}) \geq 0, \;\; \forall\, \boldsymbol{u} \in \{\boldsymbol{y} \in \mathbb{R}^p \mid \|\boldsymbol{y}\|_2 = 1\}, \;\; \forall\, \epsilon \in (0, \epsilon_{max}]. \tag{2.7}$$

This defines optimality solely based on the gradient of the loss function, without requiring the gradient to be zero at the solution and neither requiring additional second order information to be computed to ensure a local minimum solution.

NN-GPPs have been formalized on a rigorous mathematical foundation under gradient-only optimization, and proven to be equivalent to semi-positive definite local minimizers for twice continuously differentiable smooth functions [Wilke et al., 2013]. Returning to our example in Figures 2.2(a) and (b), this means that both minimization and finding NN-GPPs can be applied to the full-batch and static MBSS loss functions, where they resolve exactly the same points for unimodal functions. A whole range of gradient-only line search optimizers to locate NN-GPP have been proposed [Wilke, 2011, Wilke et al., 2013, Snyman and Wilke, 2018]. In 1-D, to locate a NN-GPP along a descent direction, merely requires a sign change in the directional derivative from negative to positive to be identified. This study recognizes that gradient-only optimization may be effective in overcoming the difficulties associated with discontinuities induced by dynamic MBSS.

However, when dynamic MBSS is employed to compute a loss function, minimization and gradient-only optimization may identify distinctly different solutions, although both definitions define the same optimal solution or true solution, $\boldsymbol{x}^{*M}$ when a unimodal full-batch sampled loss function is considered. Minimization may resolve sampling induced discontinuities local minima, whereas gradient-only optimization will largely ignore these sampling induced discontinuities unless they manifest as a sign change along a search direction. For dynamic MBSS loss functions, NN-GPP may appear and disappear stochastically in the vicinity of the true solution, $\boldsymbol{x}^{*M}$. Hence, sign changes in the directional derivative along a search direction may appear and disappear stochastically as the oracle updates the mini-batches. It is important to note that in addition to the sign change of each mini-batch loss function, additional sampling induced sign changes may manifest along a search direction. This occurs when the oracle switches between a negative and positive directional derivative for essentially the same step along a search direction. Similarly, in addition to the NN-GPP for each mini-batch loss function, additional sampling induced NN-GPP may appear and disappear in the vicinity of $\boldsymbol{x}^{*M}$. To accommodate these stochastic sampling induced NN-GPP, we extend the definition of NN-GPP to Stochastic NN-GPP (SNN-GPP) as follows:

Definition 2.1.3. *SNN-GPP: A stochastic non-negative associated gradient projection point (SNN-GPP) is defined as any point, $\boldsymbol{x}_{snngpp}$, for which there exists $\epsilon_{max} > 0$ such that*

$$\boldsymbol{u} \cdot \boldsymbol{g}(\boldsymbol{x}_{snngpp} + \epsilon \boldsymbol{u}) \geq 0, \;\; \forall\, \boldsymbol{u} \in \{\boldsymbol{y} \in \mathbb{R}^p \mid \|\boldsymbol{y}\|_2 = 1\}\,, \;\; \forall\, \epsilon \in (0, \epsilon_{max}], \tag{2.8}$$

with probability greater than 0.

There is a considerable difference in the quality of information presented in terms of proximity to true optimum when all minimizers are compared to all SNN-GPPs. Consider the red points in Figure 2.2, where we identify local minima using Definition 2.1.1 in Figure 2.2(e), and SNN-GPPs by stepping over a directional derivative sign change from negative to positive. We choose the convention of highlighting the positive directional derivative at the sign change as the SNN-GPP, as shown in Figure 2.2(f). Minimizers are largely unbounded along x, while SNN-GPPs are much more localized around the true solution x^{*M}. The majority of spurious minimizers in Figure 2.2(e), induced by dynamic MBSS, are completely ignored in Figure 2.2(f). In a recent study, a simple gradient-only line search was implemented to estimate the location of SNN-GPPs along a search direction [Kafka and Wilke, 2019]. This proved effective to automatically resolve the learning rate, and yielded competitive results when compared to probabilistic line searches [Mahsereci and Hennig, 2017]. However, the aim of this chapter is not to propose yet another algorithm, but to rather gain insight into the implication of activation functions on the nature of the loss and gradient functions, which ultimately translates to an understanding of the potential for finding SNN-GPP as opposed to minimizers in ML training.

Although many studies have been done on neural network loss functions in either a theoretical [Nguyen et al., 2018, Liang et al., 2018] or visual context [Goodfellow et al., 2015, Li et al., 2017, Im et al., 2016], most studies concentrate on the convexity properties of the expected loss function or true loss function. Given that almost all ML optimizers ignore the loss function to rather flow with the gradients [Robbins and Monro, 1951, Kingma and Ba, 2015, Duchi et al., 2011], it is sensible to rather investigate the characteristics and properties of the gradient field or directional derivatives, as well as how they are influenced by activation functions (AFs) and dynamic MBSS sampling. In our investigations, we explicitly compare static and dynamic MBSS in Section 2.2.2, and subsequently choose to focus our attention on the affects of dynamic MBSS. To the best of our knowledge this is the first study to visually explore the qualitative characteristics of gradient or directional derivative information in dynamic MBSS loss functions of neural networks. We choose a visual approach to investigate the qualitative characteristics of function value and gradient information in dynamic MBSS, such that an intuitive understanding of the relationship between minima, SNN-GPPs and activation functions [Serwa, 2017, Karlik, 2015, Laudani et al., 2015] can be developed. As we visually explore the characteristics of function values and directional derivatives for different activation functions under dynamic MBSS, it is important to continuously reassess the following: *What is the quality of function value versus directional derivative information with regards to making informed decisions on candidate solutions in mini-batch sub-sampled neural network training?*

2.1.1 Our contribution

The contribution of this work lies neither in the dataset, nor the network architecture we use. This chapter is written from the perspective where solving an optimization problem has two key components, namely a loss function landscape, and an algorithm that searches for an optima on this landscape. To take a step towards an optimum, many optimization algorithms conduct line searches, i.e. find a univariate optimum along a search direction. As discussed, mini-batch sub-sampling makes conducting line searches along search directions a non-trivial issue in loss functions with the form of Equation (2.1). Therefore, this work focusses on optimality criteria and the nature of information that is useful in static and dynamic MBSS losses. Subsequently, we wish build an intuitive and visual understanding of function minimizers and *stochastic non-negative associated gradient projection points* (SNN-GPPs) when dealing with dynamic mini-batch sub-sampled loss functions. Therefore, we restrict this chapter to an elementary neural network classification problem in order to highlight concepts with absolute clarity. The concepts explored in this chapter are not restricted to the example problem we choose, as the optimality criteria investigated apply to all loss functions, including those of state of the art deep neural networks explored by Li et al. [2017] and [Goodfellow et al., 2015]. We show that function minimization in dynamic MBSS loss functions is not effective. Instead, gradient-only optimization that finds SNN-GPPs allows for a much improved representation of full-batch optima. We demonstrate this for neural networks loss functions using the Sigmoid, Tanh, Softsign, ReLU, leaky ReLU and ELU activation functions. We also highlight some key differences between the features of the loss functions with the different activations in the context of SNN-GPPs. We show, that the characteristics of activation function's derivatives affect the localization of SNN-GPPs in weight space.

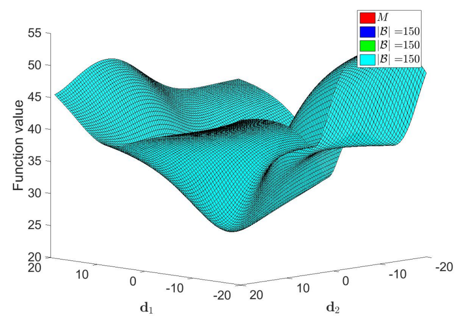
2.2 Application of concepts to a practical neural network problem

In our investigations we analyse the loss functions of a single hidden layer neural network applied to the classic Iris dataset classification problem. We use a fully connected layer containing 10 hidden units and employ the Mean Squared Error (MSE) loss. Since we do not actually train the network in our experiments, but rather investigate the loss landscape characteristics, there is no need to split the problem's dataset into training and test sets. Instead, we make all the available data available for the construction of our visualizations. The computed loss functions and directional derivative surfaces are evaluated in a 100x100 grid with range of $[-20, 20]$ units in the directions $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ around a central point, $\boldsymbol{x}_0$. This initial guess, $\boldsymbol{x}_0$, is generated from a uniform distribution with range $[-0.1, 0.1]$. Until and including Section 2.3.1, the surface plots' axes denote steps along two random, but perpendicular, unit directions [Li et al., 2017], $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ in $\mathbb{R}^p$, where $p = 83$ is the number of weights in the network. Care should be exercised in interpreting these two-dimensional visualizations of an 83-dimensional problem, as what seems to be a local optimum in these two-dimensional surfaces may well have descent directions leading away from these in different dimensions not depicted. Nevertheless, these visualizations allow the general characteristics of loss functions to be investigated. We use this platform initially to demonstrate some key concepts with the use of the Sigmoid AF, whereas from Section 2.3, we expand this to various other activation functions.

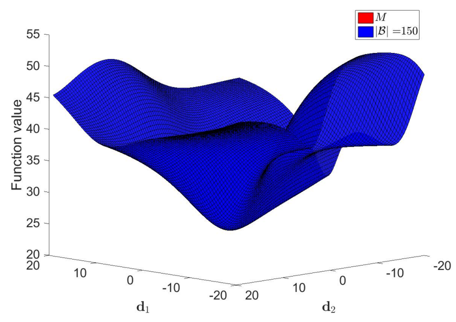
2.2.1 Full combinatorial static and dynamic mini-batch sub-sampling

Firstly, we further explore the characteristics of static and dynamic MBSS in the context of sampling uniformly from the problem dataset with replacement. Therefore, this exposes the evaluation of $\tilde{L}(\boldsymbol{x})$ to the full $K = \binom{M}{|\mathcal{B}_{n,i}|}$ possible combinations of constructing mini-batches. In Figure 2.3, we plot the resulting loss in 2-D, to gain more intuition about the observed characteristics. For all plots the true loss function $\mathcal{L}(\boldsymbol{x})$, is given in red. For static MBSS we include three different loss function expressions of $\bar{L}(\boldsymbol{x})$, whereas for dynamic MBSS we add only one plot of $\tilde{L}(\boldsymbol{x})$ in the interest of visual simplicity. We plot the function value for both static and dynamic MBSS with batch sizes $M = 150$ and $|\mathcal{B}_{n,i}| \in \{149, 10, 1\}$.

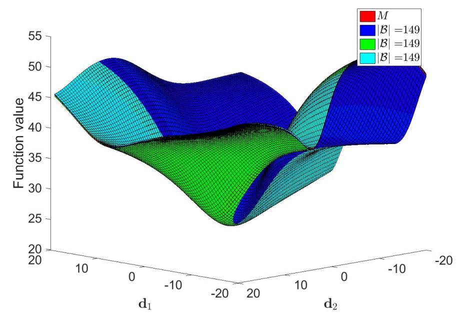
In the case where the full batch is used, Figure 2.3(a) and (b), both sampling methods are equal, as both have access to the same information at every sampled point along the given axes. When one point is omitted, i.e. $|\mathcal{B}_{n,i}| = 149$ in Figure 2.3(c), a difference begins to emerge. The individual instances of $\bar{L}(\boldsymbol{x})$ have alternative loss function curvatures to $\mathcal{L}(\boldsymbol{x})$, resulting in different patches of the loss function being visible. This becomes increasingly pronounced as the batch size is decreased. In the case of $|\mathcal{B}_{n,i}| = 10$ in Figure 2.3(e), the optima of individual $\bar{L}(\boldsymbol{x})$ differ considerably and a sampling induced offset in loss function becomes evident. It is this offset, that causes large discontinuities when alternating between batches in dynamic MBSS. If the batch size is further reduced to $|\mathcal{B}_{n,i}| = 1$ in Figure 2.3(g), the individual $\bar{L}(\boldsymbol{x})$ are no longer



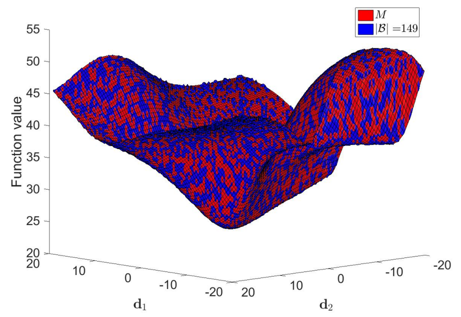
(a) $\mathcal{L}(\boldsymbol{x})$, $M = 150$



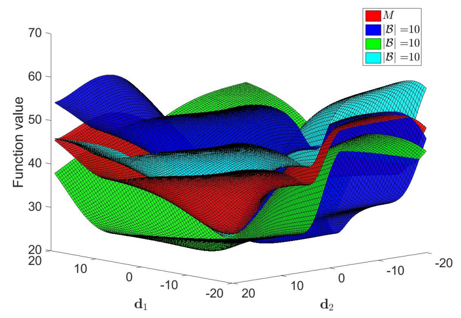
(b) $\mathcal{L}(\boldsymbol{x})$, $M = 150$



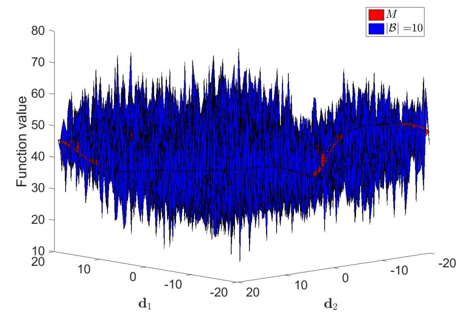
(c) $\bar{L}(\boldsymbol{x})$, $|\mathcal{B}_n| = 149$



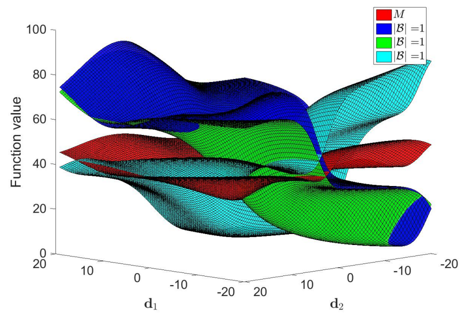
(d) $\tilde{L}(\boldsymbol{x})$, $|\mathcal{B}_{n,i}| = 149$



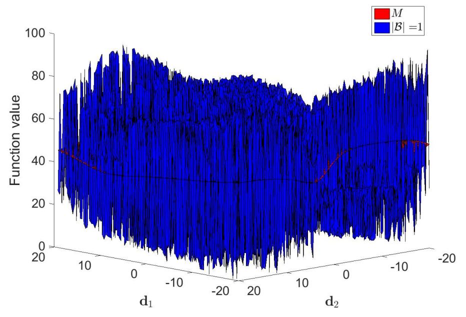
(e) $\bar{L}(\boldsymbol{x})$, $|\mathcal{B}_n| = 10$



(f) $\tilde{L}(\boldsymbol{x})$, $|\mathcal{B}_{n,i}| = 10$



(g) $\bar{L}(\boldsymbol{x})$, $|\mathcal{B}_n| = 1$

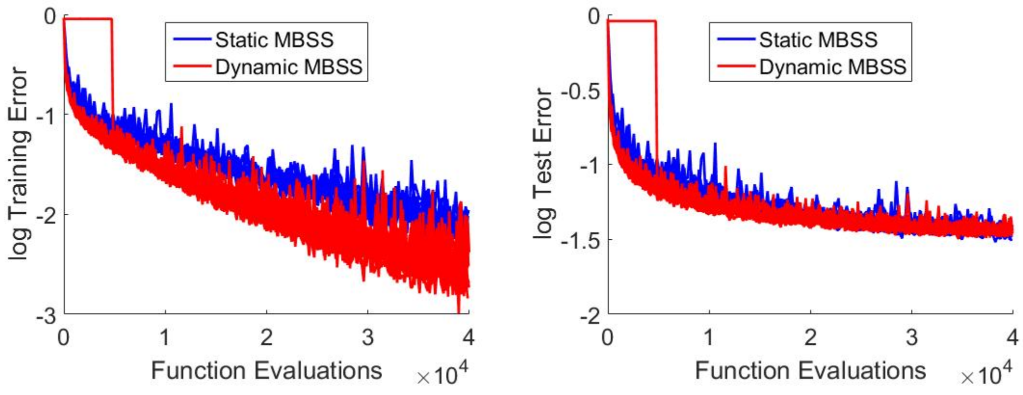


(h) $\tilde{L}(\boldsymbol{x})$, $|\mathcal{B}_{n,i}| = 1$

Figure 2.3: (a,c,e,g) Static and (b,d,f,h) dynamic mini-batch sub-sampling (MBSS) loss functions plotted along two random directions using batch sizes $M = |\mathcal{B}| = 150$ and $|\mathcal{B}| \in \{149, 10, 1\}$. Subscripts, n and i, are omitted in plot legends in the interest of compactness. A bias-variance trade-off: Static MBSS produces continuous loss functions, which are biased to the sampling error in $\mathcal{B}_n$; dynamic MBSS represents $\mathcal{L}(\boldsymbol{x})$ on average, but introduces discontinuities by re-sampling $\mathcal{B}_{n,i}$ at every loss evaluation.

reminiscent of $\mathcal{L}(\boldsymbol{x})$. Optimizing extensively in these individual loss functions would result in solutions which are not representative of $\mathcal{L}(\boldsymbol{x})$.

Dynamic MBSS on the other hand has different properties: By only omitting one random subsample at every function evaluation, the surface of $\tilde{L}(\boldsymbol{x})$ becomes discontinuous, as is evidenced by the "spotted" look of Figure 2.3(d), where blue indicates $\tilde{L}(\boldsymbol{x})$ being larger than the true or full-batch surface $\mathcal{L}(\boldsymbol{x})$ depicted in red. The severity of these discontinuities increases as the batch size decreases. However, the important aspect of dynamic MBSS is that the mean of this discontinuous plot approximates the shape of $\mathcal{L}(\boldsymbol{x})$. This indicates that the comparison between static and dynamic MBSS is akin to the bias-variance trade-off. Static MBSS gives desired characteristics for effective optimization in its smooth landscape, but is not representative of the true problem unless a large, or according to adaptive sampling methods, a representative batch is chosen. Conversely dynamic MBSS results in low bias, but high variance in the resulting loss function. Optimization algorithms which are able to operate in this mode can make use of more information in the dataset, as the information in a fixed batch size is alternated at every function evaluation. However, they need to be be able to robustly deal with the sampling induced discontinuities.



(a) Training error (b) Test error

Figure 2.4: A comparison between static and dynamic MBSS as applied in gradient-only line searches [Kafka and Wilke, 2019], showing (a) training and (b) test classification error during training of the MNIST [Lecun et al., 1998] dataset using the NetI [Mahsereci and Hennig, 2017] network architecture.

A noteworthy attempt at optimizing using dynamic MBSS has been made with the application of probabilistic surrogates (Gaussian Processes) [Mahsereci and Hennig, 2017] to conduct line searches in stochastic gradient descent (SGD). However, this method is hampered in its flexibility by having to use surrogates that have a bounded domain size for each iteration. An explicit investigation has been conducted by which static and dynamic MBSS were directly compared in the context of unbounded gradient-only line searches [Kafka and Wilke, 2019]. An example is given in Figure 2.4, where the well known MNIST [Lecun et al., 1998] dataset is trained using the NetI [Mahsereci and Hennig, 2017] network architecture. The faster decrease in both training and test classification error demonstrates that dynamic MBSS can increase the performance of training relative to computational cost. This is an example of where implementing dynamic MBSS in line searches during neural network training shows promise. In order to gain further insight, we wish to explore the landscapes of discontinuous loss functions with dynamic MBSS. Therefore, for the remainder of this chapter, dynamic MBSS is referred to when considering any form of sub-sampled loss function or directional derivative.

2.2.2 Variance properties of function value and directional derivative information in the MSE loss using dynamic MBSS

Subsequently, we explore the quality of function value and directional derivative information the context of our practical neural network problem with Sigmoid AFs. The directional derivatives were calculated using the positive diagonal, i.e. the sum of the two unit directions, given as follows:

$$\boldsymbol{d}_{dd} = \frac{(\boldsymbol{d}_1 + \boldsymbol{d}_2)}{|\boldsymbol{d}_1 + \boldsymbol{d}_2|}. \tag{2.9}$$

The directional derivative is evaluated by projecting each computed gradient onto the normalized diagonal direction $\boldsymbol{d}_{dd}$, i.e. $\tilde{D}_d(\boldsymbol{x}) = \tilde{\boldsymbol{g}}(\boldsymbol{x})^T \cdot \boldsymbol{d}_{dd}$. This allows the scalar surface of $\tilde{D}_d(\boldsymbol{x})$ to be visualized over the same 100x100 grid used plot function values. We compare the characteristics of both function value and directional derivative surfaces in Figure 2.5 with mini-batch sizes $M = 150$ and $|\mathcal{B}_{n,i}| \in \{149, 10, 1\}$.

Note, that the apparent variance in the discontinuous directional derivative surface is much lower than the discontinuous function value surface. As the batch size is reduced to $\mathcal{B}_{n,i} = 10$, the representation of $\mathcal{L}(\boldsymbol{x})$ is largely hidden in the high variance of the discontinuities in $\tilde{L}(\boldsymbol{x})$. However, in some sections of the sampled domain, the directional derivative information remains considerably more consistent with its full-batch equivalent, compared to the function value plots. This trend becomes exaggerated, as the mini-batch size is reduced further to $\mathcal{B}_{n,i} = 1$.

We formalize this observation by considering the MSE loss:

$$\ell(\boldsymbol{x};\, \boldsymbol{t}_b) = \frac{1}{2}(\boldsymbol{t}_b^o - \hat{\boldsymbol{t}}_b^o(\boldsymbol{x}, \boldsymbol{t}_b^i))^T(\boldsymbol{t}_b^o - \hat{\boldsymbol{t}}_b^o(\boldsymbol{x}, \boldsymbol{t}_b^i)), \tag{2.10}$$

where $\boldsymbol{t}_b^i$ is the b^{th} sample of the training data, $\boldsymbol{t}_b^o$ is the b^{th} output sample and $\hat{\boldsymbol{t}}_b^o(\cdot,\cdot)$ is the neural network model estimation of output $\boldsymbol{t}_b^o$ as a function of the model parameters, $\boldsymbol{x}$. and training sample, $\boldsymbol{t}_b^i$. We substitute Equation (2.10) into Equation (2.1) to obtain:

$$\tilde{L}(\boldsymbol{x}) = \frac{1}{2|\mathcal{B}_{n,i}|} \sum_{b \in \mathcal{B}_{n,i}} (\boldsymbol{t}_b^o - \hat{\boldsymbol{t}}_b^o(\boldsymbol{x}, \boldsymbol{t}_b^i))^T(\boldsymbol{t}_b^o - \hat{\boldsymbol{t}}_b^o(\boldsymbol{x}, \boldsymbol{t}_b^i)). \tag{2.11}$$

Therefore, the sampled gradient of the loss becomes

$$\tilde{\boldsymbol{g}}(\boldsymbol{x}) = \frac{-1}{|\mathcal{B}_{n,i}|} \sum_{b \in \mathcal{B}_{n,i}} (\nabla \hat{\boldsymbol{t}}_b^o(\boldsymbol{x}, \boldsymbol{t}_b^i))(\boldsymbol{t}_b^o - \hat{\boldsymbol{t}}_b^o(\boldsymbol{x}, \boldsymbol{t}_b^i)), \tag{2.12}$$

where the gradient of the model in terms of the model parameters is given by $\nabla \hat{\boldsymbol{t}}_b^o(\boldsymbol{x}, \boldsymbol{t}_b^i)$. To aid understanding of the different contributing factors, consider the notation $\boldsymbol{e}_b = (\boldsymbol{t}_b^o - \hat{\boldsymbol{t}}_b^o(\boldsymbol{x}, \boldsymbol{t}_b^i))$, which constitutes the $(q \times 1)$ prediction error vector for observation k, where q is the dimensionality of the output data and model output. The term $\boldsymbol{C}_b = \nabla \hat{\boldsymbol{t}}_b^o(\boldsymbol{x}, \boldsymbol{t}_b^i)$ denotes the $(p \times q)$ gradient matrix of the model. This simplifies Equations (2.11) and (2.12) to

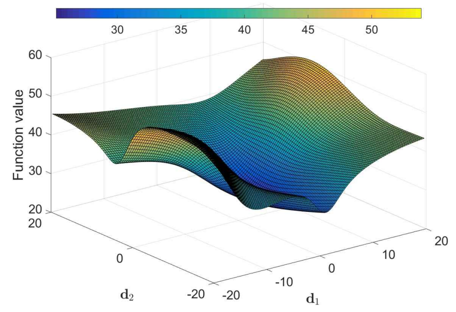
$$\tilde{L}(\boldsymbol{x}) = \frac{1}{2|\mathcal{B}_{n,i}|} \sum_{b \in \mathcal{B}_{n,i}} (\boldsymbol{e}_b^T \boldsymbol{e}_b), \tag{2.13}$$

and

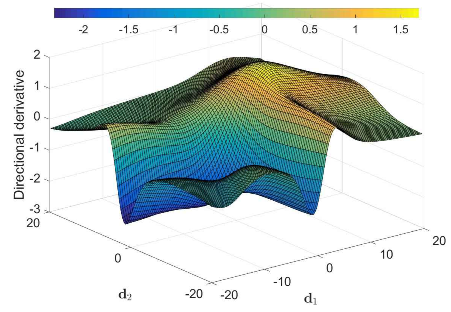
$$\tilde{\boldsymbol{g}}(\boldsymbol{x}) = \frac{-1}{|\mathcal{B}_{n,i}|} \sum_{b \in \mathcal{B}_{n,i}} (\boldsymbol{C}_b \boldsymbol{e}_b). \tag{2.14}$$

In the loss function, Equation 2.13 depends only on the $\boldsymbol{e}_b$ term, while the product of $\boldsymbol{C}_b \boldsymbol{e}_b$ determines the loss function gradient in Equation 2.14. According to the chain rule, the gradient of the model, $\boldsymbol{C}_b$, is a function of the weights, and the derivatives of the activation functions in the various layers of a neural network [Werbos, 1982]. For most activation functions, the derivative remains bounded. All the activation functions considered in our investigations have derivatives ≤ 1. This means that for a fixed batch, $\mathcal{B}_{n,i}$ and a fixed point $\boldsymbol{x}$ in space, which apply to both $\boldsymbol{e}_b$ and $\boldsymbol{C}_b$ terms, the activation function derivatives are not going to increase the magnitude of information passing through the network.

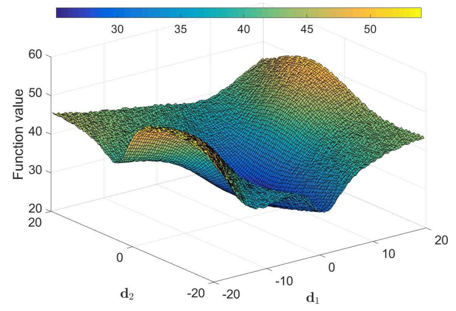
Let us now consider a fixed point in space, $\boldsymbol{x}$, where we sample many mini-batches with a constant size $|\mathcal{B}_{n,i}| < M$. In this case both $\boldsymbol{e}_b$ and $\boldsymbol{C}_b$ vary only as a function of $\boldsymbol{t}_b$, which we can separate into expected values $\bar{\boldsymbol{e}}_b$ and $\bar{\boldsymbol{C}}_b$; with corresponding variance $\sigma(\boldsymbol{e}_b)$ and $\sigma(\boldsymbol{C}_b)$ to obtain



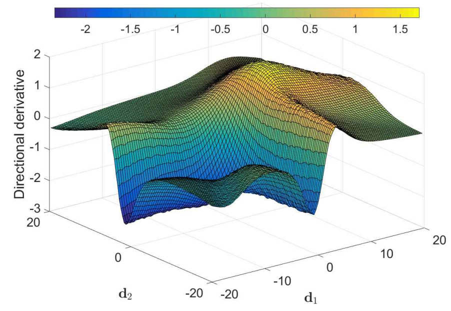
(a) $\mathcal{L}(\boldsymbol{x})$, $M = 150$



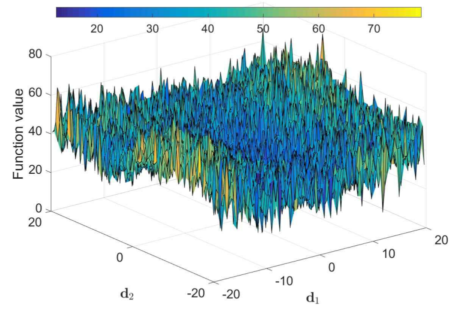
(b) $\mathcal{D}_d(\boldsymbol{x})$, $M = 150$



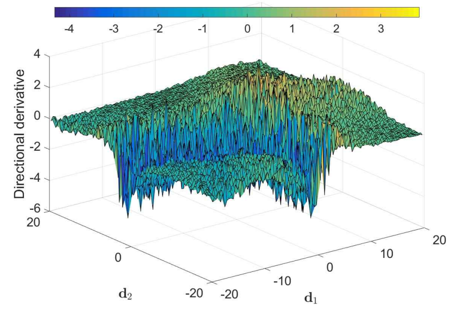
(c) $\tilde{L}(\boldsymbol{x})$, $|\mathcal{B}_{n,i}| = 149$



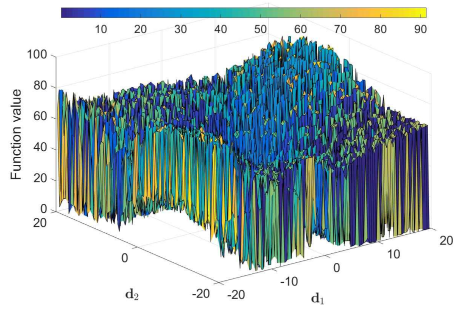
(d) $\tilde{D}_d(\boldsymbol{x})$, $|\mathcal{B}_{n,i}| = 149$



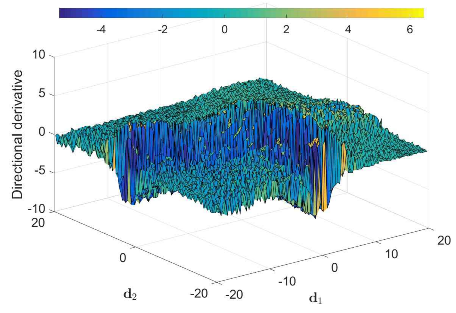
(e) $\tilde{L}(\boldsymbol{x})$, $|\mathcal{B}_{n,i}| = 10$



(f) $\tilde{D}_d(\boldsymbol{x})$, $|\mathcal{B}_{n,i}| = 10$



(g) $\tilde{L}(\boldsymbol{x})$, $|\mathcal{B}_{n,i}| = 1$



(h) $\tilde{D}_d(\boldsymbol{x})$, $|\mathcal{B}_{n,i}| = 1$

Figure 2.5: (a,c,e,g) Function value and (b,d,f,h) directional derivative plots along two orthogonal random directions, $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$, using batch sizes $|\mathcal{B}_{n,i}| \in \{150, 149, 10, 1\}$. Variance for both $\mathcal{L}(\boldsymbol{x})$ and $\tilde{D}_d(\boldsymbol{x})$ increases with decrease in batch size, with $\tilde{D}_d(\boldsymbol{x})$ being less affected than $\mathcal{L}(\boldsymbol{x})$.

$$\tilde{L}(\boldsymbol{x}) = \frac{1}{2|\mathcal{B}_{n,i}|} \sum_{b \in \mathcal{B}_{n,i}} (\bar{\boldsymbol{e}}_b + \sigma(\boldsymbol{e}_b))^T (\bar{\boldsymbol{e}}_b + \sigma(\boldsymbol{e}_b))$$

$$= \frac{1}{2|\mathcal{B}_{n,i}|} \sum_{b \in \mathcal{B}_{n,i}} (\bar{\boldsymbol{e}}_b^T \bar{\boldsymbol{e}}_b + 2\bar{\boldsymbol{e}}_b^T \sigma(\boldsymbol{e}_b) + \sigma(\boldsymbol{e}_b)^T \sigma(\boldsymbol{e}_b)), \quad (2.15)$$

and

$$\tilde{\boldsymbol{g}}(\boldsymbol{x}) = \frac{-1}{|\mathcal{B}_{n,i}|} \sum_{b \in \mathcal{B}_{n,i}} (\bar{\boldsymbol{C}}_b + \sigma(\boldsymbol{C}_b))(\bar{\boldsymbol{e}}_b + \sigma(\boldsymbol{e}_b))$$

$$= \frac{-1}{|\mathcal{B}_{n,i}|} \sum_{b \in \mathcal{B}_{n,i}} \left(\bar{\boldsymbol{C}}_b \bar{\boldsymbol{e}}_b + \sigma(\boldsymbol{C}_b)\bar{\boldsymbol{e}}_b + \bar{\boldsymbol{C}}_b \sigma(\boldsymbol{e}_b) + \sigma(\boldsymbol{C}_b)\sigma(\boldsymbol{e}_b)\right). \quad (2.16)$$

Hence, the variance in $\tilde{L}(\boldsymbol{x})$ is dictated by $2\bar{\boldsymbol{e}}_b^T \sigma(\boldsymbol{e}_b) + \sigma(\boldsymbol{e}_b)^T \sigma(\boldsymbol{e}_b)$, while the variance in $\tilde{\boldsymbol{g}}(\boldsymbol{x})$ is dictated by $\sigma(\boldsymbol{C}_b)\bar{\boldsymbol{e}}_b + \bar{\boldsymbol{C}}_b\sigma(\boldsymbol{e}_b) + \sigma(\boldsymbol{C}_b)\sigma(\boldsymbol{e}_b)$. This implies that should $|\sigma(\boldsymbol{e}_b)| >> 1$, due to changes in the mini-batch, the variance in $\tilde{L}(\boldsymbol{x})$ will be larger than the variance in $\tilde{\boldsymbol{g}}(\boldsymbol{x})$. This is due to the term $\sigma(\boldsymbol{e}_b)^T\sigma(\boldsymbol{e}_b)$ dominating in Equation (2.15), while Equation (2.16) scales linearly in both $\sigma(\boldsymbol{e}_b)$ and $\sigma(\boldsymbol{C}_b)$ which is bounded $|\sigma(\boldsymbol{C}_b)| \leq 1$, depending on $\boldsymbol{x}$.

Alternatively, the variance in $\tilde{L}(\boldsymbol{x})$ is significantly reduced when $|\sigma(\boldsymbol{e}_b)| << 1$, since

$$\sigma(\boldsymbol{e}_b)^T \sigma(\boldsymbol{e}_b) \approx 0 \quad (2.17)$$

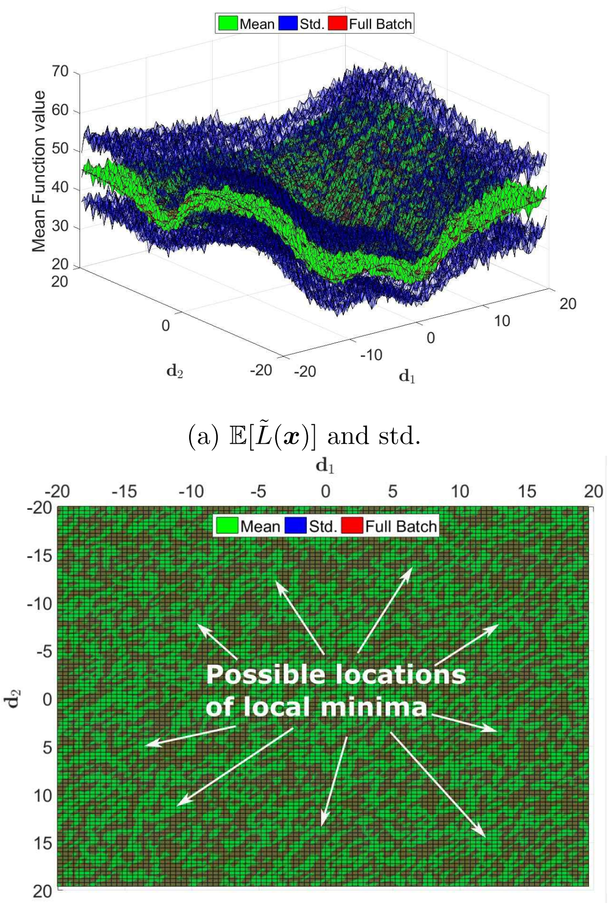
while $2\bar{\boldsymbol{e}}_b^T \sigma(\boldsymbol{e}_b)$ depends on the magnitude of $\bar{\boldsymbol{e}}_b$. In turn, the variance in $\tilde{\boldsymbol{g}}(\boldsymbol{x})$ depends mainly on $\sigma(\boldsymbol{C}_b)$ which scales $\bar{\boldsymbol{e}}_b$ and $\sigma(\boldsymbol{e}_b)$, while $\bar{\boldsymbol{C}}_b\sigma(\boldsymbol{e}_b)$ also contributes. However, since $|\bar{\boldsymbol{C}}_b| \leq 1$ and $|\sigma(\boldsymbol{C}_b)| \leq 1$ for small weights $\boldsymbol{x}$, the variance in $\tilde{\boldsymbol{g}}(\boldsymbol{x})$ is bounded. Therefore, when $|\sigma(\boldsymbol{e}_b)| << 1$ the variance in $\tilde{\boldsymbol{g}}(\boldsymbol{x})$ can be larger than that of $\tilde{L}(\boldsymbol{x})$. Also, when the magnitude of $|\bar{\boldsymbol{e}}_b| << 1$ the variance in $\tilde{L}(\boldsymbol{x})$ is reduced as can be seen in the centre of the domains in Figures 2.5(e) and (g). Lastly, it is evident that when $|\bar{\boldsymbol{C}}_b| \approx 0$, implying saturation of the activation function, the variance $|\sigma(\boldsymbol{C}_b)| \approx 0$, which in turn causes the variance of $\tilde{\boldsymbol{g}}(\boldsymbol{x})$ to diminish. The flat, low variance areas at the extremes of the sampled domains are particularly evident for the directional derivative plots sampled with small batches in Figure 2.5(f) and (h). Here, the saturation of the activation functions drives down the variance in the gradient.

2.2.3 The influence of variance on minimum and SNN-GPP optimality criteria

Subsequently, we explicitly evaluate the means and variances of function value and directional derivative surfaces by considering the same Iris dataset problem as discussed in Sections 2.2.1 and 2.2.2. The mean and variance estimates were calculated using 50 independent draws of $|\mathcal{B}_{n,i}| = 10$ samples at every one of the 100x100 grid points. In Figure 2.6(a), the full batch function value surface, $\mathcal{L}(\boldsymbol{x})$, is shown in red, which allows for the comparison between $\mathcal{L}(\boldsymbol{x})$ and the estimated mean, $\mathbb{E}[\tilde{L}(\boldsymbol{x})]$ (green). If $\mathbb{E}[\tilde{L}(\boldsymbol{x})]$ (green) dips below $\mathcal{L}(\boldsymbol{x})$ (red), given the high variance, it is likely that a local minimum may occur at the given location. Conversely, if $\mathbb{E}[\tilde{L}(\boldsymbol{x})]$ lies above $\mathcal{L}(\boldsymbol{x})$, a spurious maximum is more likely to occur.

We therefore show the function value plot from below in Figure 2.6(c), such that areas, where $\mathbb{E}[\tilde{L}(\boldsymbol{x})]$ (green) permeate through $\mathcal{L}(\boldsymbol{x})$ (red), can be easily identified. The result is a uniformly distributed mixture of red and green, indicating that local minima are spread over the entire sampled domain. Almost all of the indicated local minima misrepresent the true minimum of $\mathcal{L}(\boldsymbol{x})$. This highlights the challenges that a direct minimization strategy would face in dynamic MBSS loss functions.

When considering the directional derivative plot in Figures 2.6(b) and (d), we are concerned with localizing sign changes from negative to positive, in order to identify SNN-GPPs. Hence, we plot the zero plane in red as a visual aid for recognizing SNN-GPPs. We show Figure 2.6(b)

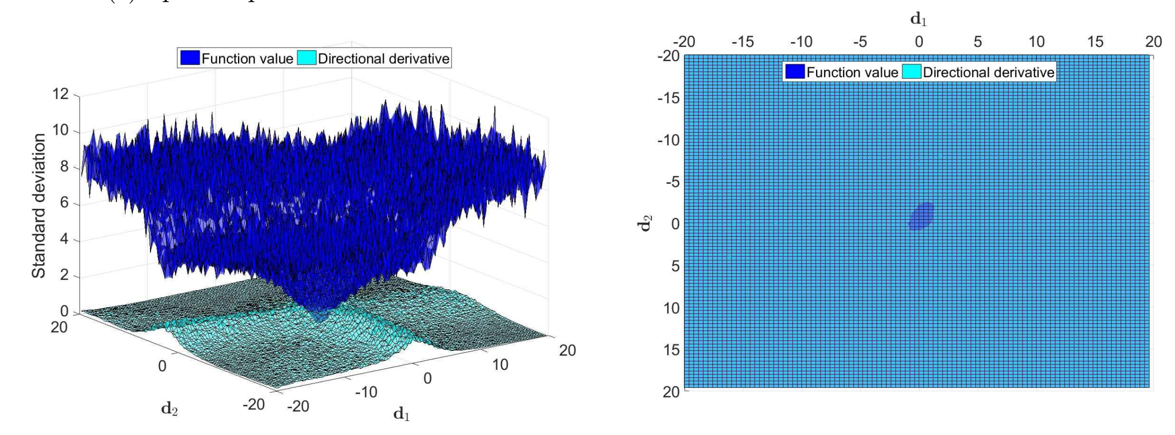


(a) $\mathbb{E}[\tilde{L}(\boldsymbol{x})]$ and std.

(b) $\mathbb{E}[\tilde{D}_d(\boldsymbol{x})]$ and std.

(c) Spatial spread of local minima

(d) Spatial localization of SNN-GPPs



(e) Side view of std. comparison

(f) Lower view of std. comparison

Figure 2.6: Estimated mean and standard deviation plots of (a) $\tilde{L}(\boldsymbol{x})$ and (b) $\tilde{\boldsymbol{g}}(\boldsymbol{x})$. In (a) $\mathcal{L}(\boldsymbol{x})$ is given in red, while in (b) the red plane denotes $d_d(\boldsymbol{x}) = 0$, to visually aid locating sign changes. Viewing these plots from below (c,d) illustrates the spatial spread of potential optima. Instances where $\mathbb{E}[\tilde{L}(\boldsymbol{x})]$ dips below $\mathcal{L}(\boldsymbol{x})$ (green areas) in (c) denote potential local minima. In (d) the green regions denote $\tilde{D}_d(\boldsymbol{x}) < 0$ and red areas $\tilde{D}_d(\boldsymbol{x}) > 0$ respectively. The white shaded areas in (d) are areas of uncertainty, where SNN-GPPs can occur. SNN-GPPs are highly localized, while minima are spread uniformly across the sampled domain. Isolating (d) $\sigma(\tilde{L}(\boldsymbol{x}))$ and (f) $\sigma(\tilde{D}_d(\boldsymbol{x}))$ surfaces, we observe a practical confirmation the discussion in Section 2.2.2.

from below (i.e. the negative domain) in Figure 2.6(d), where negative directional derivatives are presented as green and positive directional derivatives as red. The shaded domains between these two regions are uncertainty areas, where the zero plane intersects the standard deviation, but not yet the mean. Since we only depict the surface from below, the area of uncertainty is essentially double the size of that which is dark brown. Therefore, we extended this area visually with a white dotted line in Figure 2.6(d). The total encased area represents locations, where the probability of encountering a SNN-GPP is high. Immediately, there is a stark contrast between Figures 2.6(c) and (d): The areas of possible SNN-GPP occurrence are spatially localized, while the locations of possible local minima are uniformly distributed in the sampled domain. We remind the reader at this point, that both these plots represent the same problem, yet the ability to localize optima between the different optimization formulations is distinct. Searching for a solution in the form of a SNN-GPP is more representative of the true loss function, than minimizing the discontinuous loss function. The implications for optimization and line searches are clearer when considering the loss function along a 1-D search direction, as we will demonstrate in the next section.

2.2.4 Optimality criteria along 1-D search directions

As a number of optimization approaches used in neural network training make use of 1-D directional information [Robbins and Monro, 1951, Nesterov, 1983, Mahsereci and Hennig, 2017], we consider the behaviour of local minimum and SNN-GPP definitions along four selected hypothetical 1-D search directions in the 2-D sampled domain. This explores the applicability of these optimality criteria within line searches in dynamic MBSS loss functions. Note, that the euclidean norm, $\|.\|_2$ is indicated using $|.|$ from this section onwards in the interest of compactness.

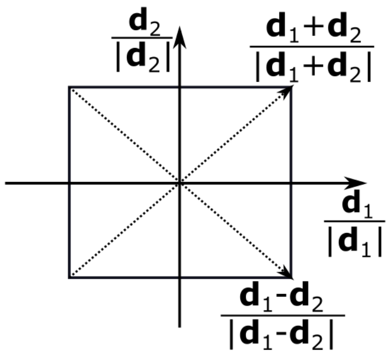

$\frac{\mathbf{d}_2}{|\mathbf{d}_2|}$
$\frac{\mathbf{d}_1+\mathbf{d}_2}{|\mathbf{d}_1+\mathbf{d}_2|}$
$\frac{\mathbf{d}_1}{|\mathbf{d}_1|}$
$\frac{\mathbf{d}_1-\mathbf{d}_2}{|\mathbf{d}_1-\mathbf{d}_2|}$

Figure 2.7: Diagram depicting the investigated 1-D directions.

The chosen directions are the unit vectors along the uncoupled axes, $\frac{\boldsymbol{d}_1}{|\boldsymbol{d}_1|}$ and $\frac{\boldsymbol{d}_2}{|\boldsymbol{d}_2|}$, as well as the diagonals of the chosen axis directions, $\frac{\boldsymbol{d}_1+\boldsymbol{d}_2}{|\boldsymbol{d}_1+\boldsymbol{d}_2|}$, $\frac{\boldsymbol{d}_1-\boldsymbol{d}_2}{|\boldsymbol{d}_1-\boldsymbol{d}_2|}$, shown in Figure 2.7. These directions are arbitrarily chosen in order to explore examples of univariate loss function characteristics that might be encountered in line searches. Typically, these directions are given by an optimization algorithm. If the domain range of the 2-D plots is $[r_{min}, r_{max}]$ in both $\frac{\boldsymbol{d}_1}{|\boldsymbol{d}_1|}$ and $\frac{\boldsymbol{d}_2}{|\boldsymbol{d}_2|}$, we sample $\tilde{L}(\boldsymbol{x})$ and $\tilde{D}_d(\boldsymbol{x})$ in $n = \{1, 2, \dots 100\}$ discrete steps, $\alpha = \frac{r_{max}-r_{min}}{n_{max}}$, along $\frac{\boldsymbol{d}_1}{|\boldsymbol{d}_1|}$ and $\frac{\boldsymbol{d}_2}{|\boldsymbol{d}_2|}$ and correspondingly, $\alpha = \sqrt{2} \cdot \frac{r_{max}-r_{min}}{n_{max}}$ along $\frac{\boldsymbol{d}_1+\boldsymbol{d}_2}{|\boldsymbol{d}_1+\boldsymbol{d}_2|}$ and $\frac{\boldsymbol{d}_1-\boldsymbol{d}_2}{|\boldsymbol{d}_1-\boldsymbol{d}_2|}$, while exhaustively counting the number of local minimizers and SNN-GPPs along each direction. Minima are counted using Definition 2.1.1, by comparing the function values of 3 adjacent discrete points, $\tilde{L}(\boldsymbol{x}_n)$, $\tilde{L}(\boldsymbol{x}_{n+1})$ and $\tilde{L}(\boldsymbol{x}_{n+2})$ respectively. If $\tilde{L}(\boldsymbol{x}_n) > \tilde{L}(\boldsymbol{x}_{n+1}) < \tilde{L}(\boldsymbol{x}_{n+2})$, a local minimum is identified and counted. SNN-GPPs are counted when a sign change in directional derivative $\tilde{D}_d(\boldsymbol{x})$ from negative, i.e. $\tilde{D}_d(\boldsymbol{x}_n) < 0$, to positive, i.e. $\tilde{D}_d(\boldsymbol{x}_{n+1}) > 0$, is observed, along the search direction. The SNN-GPP is indicated at the positive sample, $\tilde{D}_d(\boldsymbol{x}_{n+1})$, for consistency sake. The total number of candidate optima along all four directions is counted and given in the legend of the plots. We also show histograms of the spatial distribution of the minimizers and SNN-GPPs along the respective search directions.

The true optima of the 1-D curves are indicated by the full batches in Figure 2.8(a) and (b), where we see equivalence between true minima and true NN-GPPs for each respective search direction. As was the case in the 2-D plots of Figure 2.6, when dynamic MBSS is implemented,

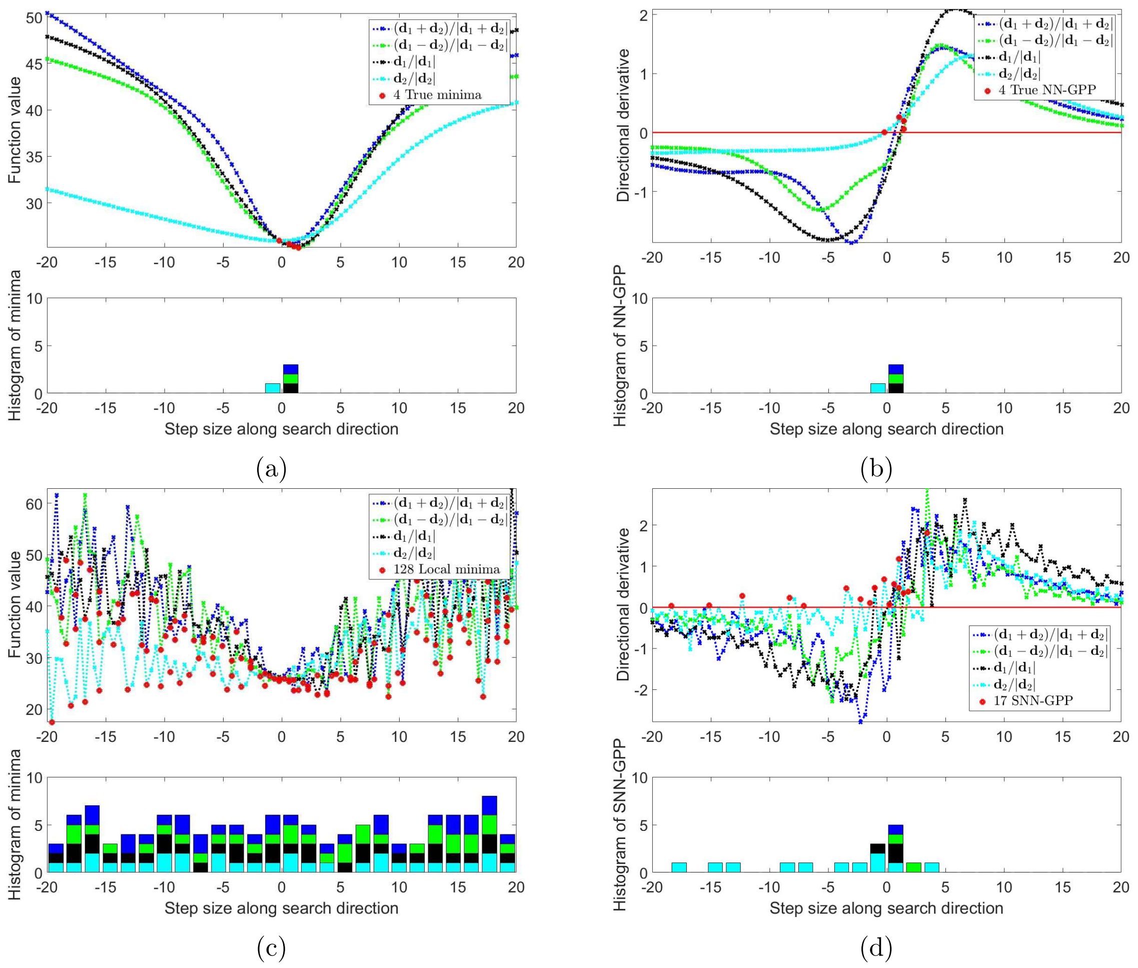


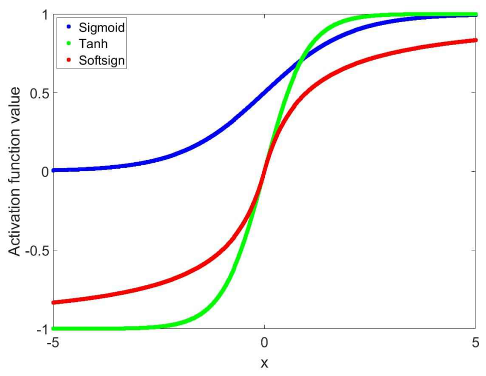
Figure 2.8: 1-D plots of the (a) function value and (b) directional derivative along fixed directions of the 2-D domains. We compare two states, using the full-batch loss with $M = 150$, and dynamic MBSS loss with, $|\mathcal{B}_{n,i}| = 10$. We note the number local minima and SNN-GPP in either case according to the respective optimality criteria. Again, the locations of the local minima are uniformly distributed along the sampled domains for all directions. In the case of the directional derivative, directions that contain high curvature, such as $\boldsymbol{d}_1$, have highly localized sign changes. Direction $\boldsymbol{d}_2$ progresses along the area of uncertainty in Figure 2.6 for some of its domain, which makes the location of SNN-GPP more widespread.

the spatial distribution of local minima is spread uniformly along the search directions, as shown in Figure 2.8(c). The total number of local minima counted over the 4 search directions is 128. This means that a minimization line search is exposed to 124 potential false candidate solutions across the chosen search directions. Arguably, the sensitivity of a minimization line search to the number of candidate solutions depends on its particular algorithmic formulation [Arora, 2011]. A significant probability exists, that a discontinuity occurs at a given loss function evaluation in dynamic MBSS. Therefore, every increment (step) within a line search poses the risk of exposing a spurious minimum. However, not all local minima might be accepted by a line search, especially when inexact line searches are conducted. The well known Armijo and Wolfe conditions [Arora, 2011], as often used in inexact line searches, provide rules which can reduce the range of candidate minima to be accepted. Nevertheless, the local minimum definition provides a large number of candidate optima within a given formulation's acceptable range, which the particular formulation is required to contend with, reducing its effectiveness Kafka and Wilke [2019b]. In turn, the spatial distribution of SNN-GPPs is much more localized around the true optima as shown in Figure 2.8(d). The direction with the highest number of SNN-GPPs is $\boldsymbol{d}_2$, since the first half of $\boldsymbol{d}_2$ runs along an area of uncertainty for SNN-GPP, seen in Figure 2.6(d). Due to the low magnitude in directional derivative and the correspondingly high variance in the sampled gradient values, it is possible to encounter spurious sign changes along this ridge. The remaining three directions show SNN-GPPs to be are tightly clustered around the true optimum, indicating small spatial domains of uncertainty. This means that a line search strategy that looks for SNN-GPPs would be significantly more effective in approximating the location of the true optimum.

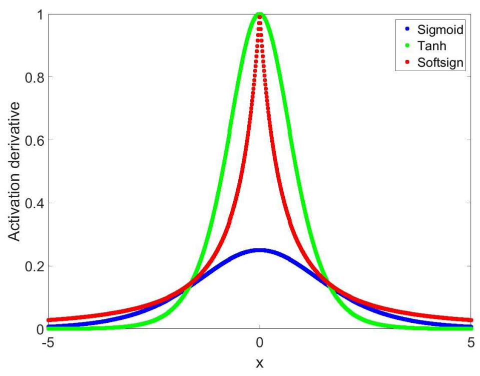
Subsequently, we use the tools developed to investigate the Sigmoid AF, to extend our consideration to a number of popular activation functions, investigating their effect on finding minimizers and SNN-GPPs as well as observing their variance around the true optimum along a search direction.

2.3 Alternative Activation Functions

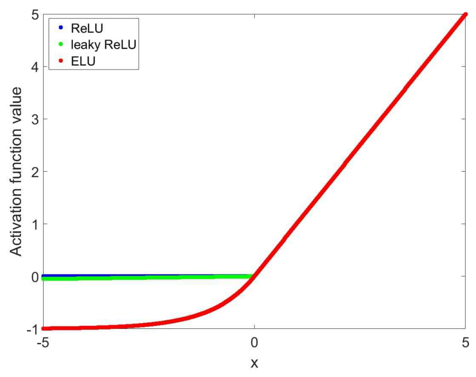
Though Sigmoid activations were popular in the past, other smooth activations, such as Tanh [Karlik, 2015] and Softsign [Bergstra et al., 2009], are often preferred. Collectively, this type of AF has been called the saturation class of AFs [Xu et al., 2016]. Today, the state of the art networks use AFs that promote sparsity, namely the ReLU [Glorot and Bordes, 2011] family of activations. We include leaky ReLU [Maas et al., 2013] and ELU [Clevert et al., 2016] into this category and refer to this family as the sparsity class of AFs. Both these classes are plotted in Figure 2.9 for comparison.



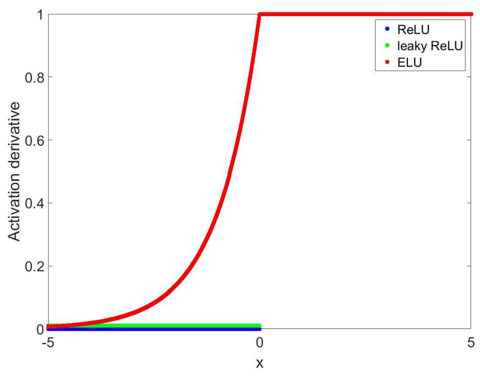
(a) Saturation class function values



(b) Saturation class derivatives



(c) Sparsity class function values



(d) Sparsity class derivatives

Figure 2.9: (a,c) Function value and (b,d) derivatives of activation functions considered in our investigations. These are grouped together into (a,b) saturation and (c,d) sparsity classes respectively. The saturation class makes use of the non-linearities of the activation functions to construct mappings. The sparsity class makes use of the architecture in the network to introduce the non-linearities, where specific "channels" in the network are selectively activated, depending on whether the activation function transmits information or not.

As their name suggests, the dominant characteristic of the saturation class functions is the stagnating, plateau-like behaviour at in both negative and positive extremes of the x-domains.

The differences within the saturation class concentrate primarily on the range over which the transition from saturation to the active domain (where the gradient is high) and back to saturation occurs. Activation functions with small "active" ranges are noted as having high saturation characteristics, while those that have higher gradients over a larger domain are said to have low saturation characteristics. The active domains are centred around $x = 0$, but not all run through the origin. The Sigmoid AF is only positive, while Tanh and Softsign pass through the origin and allow both negative and positive outputs. As we will demonstrate, the differences between the activation functions, though slight in their elemental form, can lead to significant changes in loss function characteristics.

The sparsity class have fundamentally different characteristics to the saturation class in that they allow a linear relationship in their "activated" positive domain, while enforcing or approximating an "off" characteristic in the negative domain. The ReLU activation function enforces a "hard off", setting the function value and derivative to 0 for the negative x-domain. The leaky ReLU allows a small constant positive derivative in the negative domain of x. This allows a small amount of information to permeate the network when the leaky ReLU is in its "off" state. The transition to the positive domain x is discontinuous in the derivative, which like ReLU enforces a drastic change in characteristics within the loss function. The ELU formulation constructs the derivative to be smooth and continuous, in the form of an exponential, when transitioning from negative to positive along x.

2.3.1 Global activation function characteristics in random directions over [-20,20] domains in weight space

We now investigate the effect of the above discussed AFs on the distribution of minimizers and SNN-GPPs. To allow for a direct comparison between AF-related loss function characteristics, we use the same problem and architecture as explored with Sigmoids in Section 2.2, with the only change being the AFs used in the neural network. The results are summarized in Table 2.1. From Table 2.1 it is evident that the number of SNN-GPPs is much closer to that of the true local minima, in comparison to the number of local minimizers for all activation functions. In general, there are 12 to 32 times more identified local minimizers than true optima, whereas there are only 1.5 to 6 times more SNN-GPPs than true optima. The number SNN-GPPs is the closest to that of the true optima for the Softmax and ELU AFs, while Sigmoid and Tanh AFs resulted in the largest estimation of SNN-GPPs compared to true optima.

The spatial distribution of the minimizers and SNN-GPPs are given in Appendix A.1.1 as histogram plots. In the histogram plots, since SNN-GPPs are indicated as being after the sign change in our analyses, cases can occur where the location of the true optimum is close to edge of a bin in the histogram. In such instances the location of the SNN-GPP might be noted in the bin that is adjacent of that indicated for the minima. This is not an error, but simply an artefact of how local minima and SNN-GPP are identified from discrete samples of the loss function and directional derivatives respectively.

One of the prominent differences between the saturation and sparsity activation function classes is the behaviour on the extremities of the sampled domains. The saturation functions have "mountainous" features, in the sense that the landscape alternates between high and low curvature regions. This occurs as different nodes in the network saturate, or activate. When most but not all nodes have saturated, flat planes with low directional derivative values can be observed. This can have a negative effect on the spatial distribution of SNN-GPPs, as seen in Figures 2.8(d) and A.1(h). In cases where saturation is high (the AF derivative tends quickly towards 0, as $x \to \pm\infty$, Figure 2.9(b)), the directional derivative magnitude is low. Hence, the variance of the directional derivative is lower than other areas of the loss function, but due to the small directional derivative magnitude, it may still create spurious SNN-GPPs. This is particularly prevalent for the Tanh activation function, see Figure A.1(h). Conversely, SNN-GPPs are highly localized for the Softsign analysis, since saturation occurs much further from $x = 0$, with higher derivatives at the edges of the domain, see Figures 2.9(b) and A.2(h). This demonstrates that the derivative shape of the activation function matters with respect to finding

AF	Figure	# True Optima	# Local Minima	# SNN-GPPs	Comments
Sigmoid	2.5, 2.8	4	128	17	Smooth features, spurious SNN-GPPs in low curvature directions.
Tanh	A.1	4	126	23	Higher curvature; SNN-GPPs more localized, but high saturation, leading to spurious SNN-GPPs at edges.
Softsign	A.2	4	117	6	More curvature than Sigmoid, less than Tanh; more curvature at origin gives better localization; less saturation at ends means less spurious SNN-GPPs at edges.
ReLU	A.3	8	104	13	Exponential behaviour in MSE loss; Globally convex shape, but locally intricate; more true optima; no saturation, no spurious SNN-GPPs at edges.
Leaky ReLU	A.4	8	101	17	No different to ReLU at global scale; low curvature directions have more spurious SNN-GPP due to leaky gradient (as opposed to hard-zero).
ELU	A.5	4	99	6	Less true optima than other ReLUs; smooth loss function features; higher gradient curvature, resulting in more localized SNN-GPPs.

Table 2.1: Summary of observations from analyses conducted by traversing random directions over a scalar range of [-20,20] to give an impression of ”global” loss function characteristics. The corresponding Figures A.1-A.5 are included in Appendix A.1.1.

SNN-GPPs.

The sparsity class exhibits very steep behaviour at the outer limits of the domain. This is a feature related to the ”on” nature of the activation functions. When most/all activations of the network nodes are in the positive domain, the input data is passed through the network and potentially amplified by larger weights. Since we implement the MSE loss for these examples, any classification error above 1 gets amplified, resulting in the aggressive increase in error. The consequence of this is a relatively convex looking loss function on a ”global” scale. However, in the centre of the domain, where the error is small, a significant amount of detail is present, that is lost at this scale. It is also notable, that the number of true optima obtained when using ReLU and leaky ReLU AFs are different to the rest for the same problem. This is a clear indication, that there are unique features closer to the origin, that distinguish these activations from the rest. In search for these features, we conduct a second analysis over a more local domain in the weight space in Section 2.3.2.

2.3.2 Local activation function characteristics in descent directions over [-2,2] domains in weight space

In this section we modify the definitions of $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$, to be the steepest descent direction of the full batch, $\boldsymbol{d}_1 = -\nabla\mathcal{L}(\boldsymbol{x})$ at initial starting point $\boldsymbol{x}_0$ located at (0,0), and $\boldsymbol{d}_2$ is chosen to be a random direction perpendicular to $\boldsymbol{d}_1$. We also reduce the grid range to [-2,2], to give a more detailed perspective of characteristics around local minima for different AFs. A summary of the observations made is given in Table 2.2.

In this case, the number of local minima counted is 21 to 33 times higher than the number of true optima present in the loss function. In comparison, the equivalent occurrence of SNN-

GPPs compared to true optima is a factor of 3 to 9. Due to the smaller relative domains, the step sizes along respective search directions are smaller, creating a higher chance of SNN-GPPs occurring close to a true optimum. This accounts for the slight increase in the ratio between counted SNN-GPPs and true optima, compared to the larger [-20,20] domains. The AF resulting in the least identified SNN-GPPs for the saturation and sparsity classes are Softsign and ELU respectively, while the corresponding worst performers are Tanh and leaky ReLU.

AF	**Figure**	**# True Optima**	**# Local Minima**	**# SNN-GPPs**	**Comments**
Sigmoid	A.6	4	117	36	Smooth features; spurious SNN-GPP again in low curvature directions; mainly the contour direction.
Tanh	A.7	4	130	15	Higher curvature than Sigmoid; SNN-GPP fewer, more localized; contour direction generates spurious SNN-GPP over whole domain.
Softsign	A.8	4	123	13	More local curvature than Tanh; SNN-GPP highly localized.
ReLU	A.9, A.12	5	105	17	Piece-wise, multi-modal with shallow basins; directional derivatives are discontinuous; existence of flat planes when all nodes are "off" (insensitive to sub-sampling); quadratic when all nodes are "on"
Leaky ReLU	A.10, A.12	5	124	17	Similar features to ReLU, but "flat planes" have slight curvature
ELU	A.11	4	121	11	Smooth loss function features; unimodal in diagonal directions; continuous directional derivative; higher gradient curvature; most localized SNN-GPP for ReLUs.

Table 2.2: Summary of observations from analyses conducted by traversing a scalar range of [-2,2] to give an impression of "local" loss function characteristics. Here the search directions are $\boldsymbol{d}_1 = -\nabla\mathcal{L}(\boldsymbol{x})$ and $\boldsymbol{d}_2$ is a random perpendicular direction to $\boldsymbol{d}_1$. The corresponding Figures A.6-A.11 are shown in Appendix A.1.2.

Considering specific AFs, the most prominent feature of the ReLU and leaky ReLU loss functions is that they do not have uniform characteristics. As is shown in Figure A.9(a), there are flat planes, low curvature optima, high curvature optima, and quadratic characteristics almost seemingly "stitched" together. There are up to two optima in a given search direction. Some are in very shallow basins (in the negative domains) and others are in basins of higher curvature (around 0.5 units). This "stitched" behaviour in the loss function are a result of the discontinuous nature of the AF derivative, seen in Figures A.9(b) and A.10(b). Here, the steps indicate different nodes switching "on" and "off". The less nodes are active, the less curvature is present in the loss function. Conversely, if many nodes are simultaneously "on" with high weight values, the loss function increases quadratically. This explains the multi-modality in a given search direction, which is in stark contrast to the other activation functions, that are smooth and continuous.

Importantly, there are also domains in the ReLU loss function where the directional derivatives are exactly 0. These are domains where all nodes are "off" and no information is able to pass through the network. These areas are unaffected by mini-batch sub-sampling, exhibiting directional derivatives that are consistently 0, see Figure A.12(a). This occurs when the network weights are such that all the incoming information from the data is pushed into the negative domain of the ReLU activation function. These are problematic areas for neural network training, since most training algorithms make use of gradient information to update $\boldsymbol{x}$. If there is no gradient information, there will be no updates to the state of the network and training comes to a premature halt. These flat planes are also present for the leaky ReLU. However, these do not have derivatives that are 0 (see Figure A.12), thus preventing training algorithms from stagnating during optimization. The additional curvature and smooth transition of gradients given by the ELU activation function further aid in constructing smooth features in the loss function. In this analysis these characteristics have led to a lower number of local optima, and have further aided the location of SNN-GPP in dynamic MBSS loss functions (see Figure A.11).

For the saturation class of AFs, we see similar trends in this analysis compared to that of the "global" domains: High curvature directions contain fewer and more localized SNN-GPPs. Low curvature directions result in a wider range of possible SNN-GPP locations and in some cases can even cause the likely location of SNN-GPP to shift. Specifically, the close-up Softsign analysis shows the disappearance of a SNN-GPP out of the edge of the sampled domain. The true NN-GPP at step size around ± -1.9 moves beyond [-2,2] to fall between [-3.5,-3.1] (confirmed by multiple mini-batch analyses in this extended domain). This SNN-GPP is far from the true NN-GPP. If an optimization method were to find the solution in the [-3.5,-3.1] range, it would be a poor representation of the true optimum, though the consequence in terms of error from the model might be low, since the change in error is not high around this area. However, this underlines the argument of preferring directions of high curvature for optimization purposes.

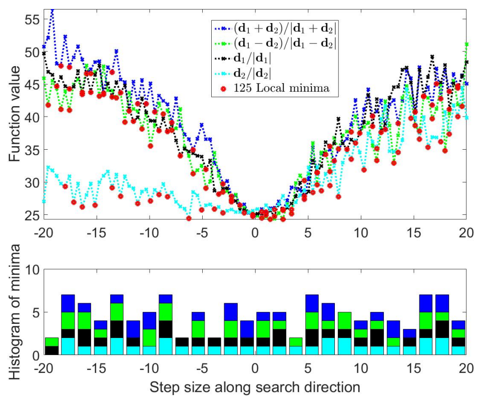
Overall, the same general trend holds for investigations on both the global and local domains conducted in this chapter: The characteristics of the optimality formulations explored in Section 2.2.2 hold across all considered activation functions. The number of local minima counted is often around an order of magnitude higher than the equivalent number of SNN-GPPs in dynamic MBSS loss functions. Local minima are spatially spread across the whole domain, whereas only low curvature directions exhibit a larger number of spurious SNN-GPPs. Due to their favourable derivative characteristics, the activation functions that produced the best results for finding SNN-GPPs were Softsign and ELU for their respective classes. Smooth derivatives and high curvature directions are concepts that align well with attempts to incorporate mathematical programming [Arora, 2011] methods into machine learning [Schraudolph et al., 2007, Martens, 2010].

2.4 Sensitivity of optimum formulations to mini-batch sample size

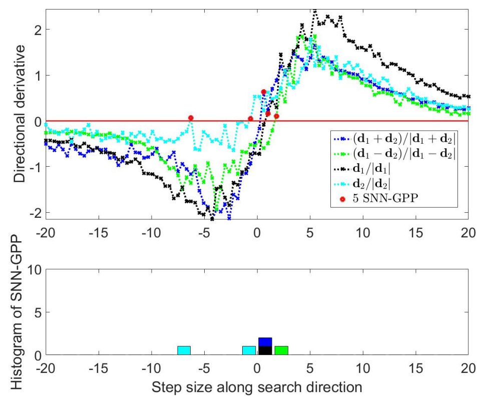
Through our analyses of different activation functions with a constant batch size of $|\mathcal{B}_{n,i}| = 10$, we have demonstrated the sensitivity of optimality criteria to curvature in the loss function. To round off the discussion around different optimum formulations, we show a representative example of sensitivity to batch size. We again use the Sigmoid AF (the worst performer in the saturation class) and conduct the same analyses with domains and directions as shown Sections 2.3.1 and 2.3.2, only changing the mini-batch sample size to $|\mathcal{B}_{n,i}| = 50$.

The results shown in Figure 2.10(b) and (d) immediately exhibit a lower occurrence and smaller spatial localization of SNN-GPP, especially in the low-curvature direction, $\boldsymbol{d}_2$, compared to the results in Figures 2.8(d) and A.6(h). The increase in sample size reduces the variance in the discontinuities due to changing mini-batches, which affects both function values and directional derivatives. This is particularly effective for increasing the robustness of locating SNN-GPPs in low curvature domains.

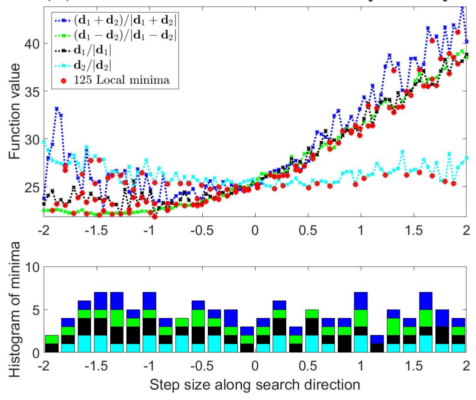
We compare the quantitative data of analyses with $|\mathcal{B}_{n,i}| = 10$ and $|\mathcal{B}_{n,i}| = 50$ in Table 2.3. In both the global and local domains, the number of SNN-GPPs counted dropped by roughly a factor of 3. With regards to function values, the number of local minima remained similar to those counted for $|\mathcal{B}_{n,i}| = 10$. This indicates that while local minima are largely insensitive to a decrease in loss function variance, SNN-GPPs react positively to a reduction in gradient



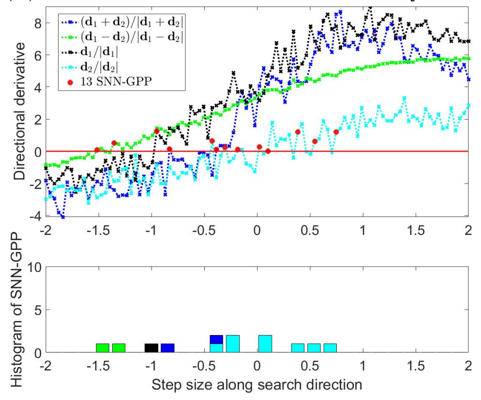
(a) Function value, domain [-20,20]



(b) Directional derivative, domain [-20,20]



(c) Function value, domain [-2,2]



(d) Directional Derivative, domain [-2,2]

Figure 2.10: Function values and directional derivatives along fixed search directions using mini-batch size $|\mathcal{B}_{n,i}| = 50$. The increase in batch size reduces the severity of sampling error. This is particularly effective in low curvature directions, causing the locations of SNN-GPPs to be more consistent. However, the localization of local minima is not positively affected by this increase in $|\mathcal{B}_{n,i}|$.

variance. As the sample size increases, the number and location of SNN-GPPs more closely approximates that of the true optima. However, as demonstrated by Figure 2.5, this does not occur as rapidly with the increase of mini-batch size for the local minima formulation. We can only say with certainty that finding local minima is only reliable in the limit case of using all data, or keeping the mini-batch static. The use of SNN-GPPs offers an improved alternative to estimate the solution, in particular, when loss functions are computed using dynamic MBSS with smaller batch sizes.

domain	**# True Optima**	**# Local Minima for $|\mathcal{B}_{n,i}| = 10$**	**# Local Minima for $|\mathcal{B}_{n,i}| = 50$**	**# SNN-GPPs for $|\mathcal{B}_{n,i}| = 10$**	**# SNN-GPPs for $|\mathcal{B}_{n,i}| = 50$**
[-20,20]	4	128	125	17	5
[-2,2]	4	117	125	36	13

Table 2.3: Number of local minima and SNN-GPPs counted in different domain sizes as a function of mini-batch size, $|\mathcal{B}_{n,i}|$.

2.5 Conclusion

This chapter visually explores the local minimum and stochastic non-negative associated gradient projection point (SNN-GPP) definitions in the context of dynamic mini-batch sub-sampled

(MBSS) loss functions of neural networks with different activation functions. We have highlighted the differences between static and dynamic MBSS strategies and have linked their properties to the bias-variance trade-off respectively. Static sub-samples result in a smooth loss function, which is beneficial to minimization optimization methods, but results in a bias towards the selected mini-batch. Conversely, dynamic MBSS results in an unbiased but high variance loss function with discontinuities, which obstructs minimization based optimizers. However, the fact that new information is presented at every cycle in dynamic MBSS, seems to have performance benefits to an optimization method that is able to operate in discontinuous loss functions.

We investigate the ability of local function minimizers and SNN-GPPs to localize true (full-batch) optima when dynamic MBSS is implemented. Function minimizers formulated as the standard mathematical programming problem give rise to a large number of spurious local minima, which are high in number and uniformly distributed throughout the domain. This may significantly hamper the effectiveness of minimization line searches in neural network training using both exact and inexact line searches. Although inexact line searches may overcome some minima, it is not guaranteed, especially when monotonic descent is enforced to achieve convergent line search strategies.

SNN-GPPs, formulated as the solution to the gradient-only optimization problem, have been shown to be spatially localized around the true optimum. Our results have shown that SNN-GPPs have on average an order of magnitude less chance of occurrence than local minima over the same domain. With increasing mini-batch sub-sample sizes, the chance of spurious SNN-GPPs decreases even further, while local minima remain spatially spread. This holds for both "global" and "local" investigations performed. The spatial variance of SNN-GPPs depends on the curvature in the loss function along the search direction. High curvature tends to result in spatially concentrated SNN-GPPs, while low curvature search directions result in larger areas of uncertainty. The curvature of the loss function is sensitive to the activation function chosen in the neural network. In the problem considered, we showed that it is of interest to choose activation functions that result in high curvature in the directional derivative, while avoiding characteristics that might lead to the construction of numerous optima, or flat planes with zero derivatives.

We divide the investigated activation functions into two classes, namely: Saturation and sparsity. These classes have different characteristics and therefore can be selected based on requirements of the given problem. According to our investigations, the activation functions for each class with the most favourable characteristics for locating high quality SNN-GPPs in dynamically sub-sampled loss functions were Softsign and ELU respectively. The ability to find optima more reliably in dynamic MBSS loss functions allows for exploration in constructing better optimizers for memory-restricted machine learning applications. Neural network models in particular can benefit from considering training as finding SNN-GPP as opposed to minimizing the loss function. This perspective of optimization may improve the construction of efficient and effective line searches in dynamic MBSS losses, which is still an open problem.

Chapter 3

Resolving learning rates adaptively by locating SNN-GPPs using line searches

Learning rates in stochastic neural network training are currently determined *a priori* to training, using expensive manual or automated iterative tuning. Attempts to resolve learning rates adaptively, using line searches, have proven computationally demanding. Reducing the computational cost by considering mini-batch sub-sampling (MBSS) introduces challenges due to large variance in information between batches that may present as discontinuities in the loss function, depending on the MBSS approach. This chapter proposes a robust approach to adaptively resolve learning using *dynamic* MBSS loss functions. This is achieved by finding sign changes from negative to positive along directional derivatives, which ultimately converge to a stochastic non-negative associated gradient projection point (SNN-GPP). Through a number of investigative studies we demonstrate that this gradient-only line search (GOLS) approach allows the learning rate to be adaptively resolved, convergence performance to improve over minimization line searches, some local minima ignored, and an otherwise expensive hyper-parameter eliminated. We also show that poor search directions may benefit computationally from overstepping optima along a descent direction, which can be resolved by considering improved search directions. Having shown that GOLS is a reliable line search, for the first time we can investigate the benefits of *dynamic* over *static* MBSS.

3.1 Introduction

The aim of this chapter is to compare gradient-only line searches to minimization line searches in the context of mini-batch sub-sampled loss functions, that result in discontinuous loss functions often encountered in Artificial Neural Network training. Given the ability to conduct line searches in this environment, it would be possible to adaptively and robustly resolve step sizes (learning rates). This chapter presents a formulation to achieve this. And to this end, we introduce the various concepts required to conduct our investigations.

3.1.1 Loss Functions and Gradients

Machine learning models are often posed with loss functions of the form

$$\mathcal{L}(\boldsymbol{x}) = \frac{1}{M}\sum_{b=1}^{M} \ell(\boldsymbol{x};\ \boldsymbol{t}_b), \tag{3.1}$$

where $\boldsymbol{x} \in \mathbb{R}^p$ is a p-dimensional vector of parameters, $\{\boldsymbol{t}_1, \ldots, \boldsymbol{t}_M\}$ is a training set of M data points of dimension D, and $\ell(\boldsymbol{x};\ \boldsymbol{t})$ is an elemental loss function quantifying the performance of parameters $\boldsymbol{x}$ on training sample $\boldsymbol{t}$. The exact gradient w.r.t. $\boldsymbol{x}$ is then computed by back-propagation [Werbos, 1994], giving

$$\nabla\mathcal{L}(\boldsymbol{x}) = \frac{1}{M}\sum_{b=1}^{M}\nabla\ell(\boldsymbol{x};\ \boldsymbol{t}_b). \tag{3.2}$$

For large M, the exact gradient can become computationally demanding to compute. Instead, a mini-batch sample, $\mathcal{B} \subset \{1,\ldots,M\}$ of size $|\mathcal{B}| \ll M$ is selected from the training set to compute an approximate loss function

$$L(\boldsymbol{x}) = \frac{1}{|\mathcal{B}|}\sum_{b\in\mathcal{B}}\ell(\boldsymbol{x};\ \boldsymbol{t}_b), \tag{3.3}$$

and approximate gradient

$$\boldsymbol{g}(\boldsymbol{x}) = \frac{1}{|\mathcal{B}|}\sum_{b\in\mathcal{B}}\nabla\ell(\boldsymbol{x};\ \boldsymbol{t}_b). \tag{3.4}$$

Here, $\mathcal{L}(\boldsymbol{x})$ refers to a true or full-batch loss function and $L(\boldsymbol{x})$ to an approximate or sub-sampled loss function. It is important to note that since we compute $L(\boldsymbol{x})$ using only a subset of the training dataset, we introduce a sampling error. This sampling error can manifest in various ways in the computed loss function, when conducting mini-batch sub-sampling (MBSS). This mainly depends on i) *when* during optimization the mini-batch is updated from the training data set, ii) the number of samples drawn per mini-batch and iii) *how* the samples are selected for a mini-batch from the training data set.

This investigation is primarily concerned with the implications of i) *when* the mini-batch is updated from the training data set. First, we consider the standard approach to update a mini-batch, which is at the beginning of every unit search direction $\boldsymbol{u}$ [Bollapragada et al., 2017, Friedlander and Schmidt, 2011, Martens, 2010]. The mini-batch is kept fixed along a search direction and only newly sampled once a new search direction is computed. We refer to this mini-batch sampling approach as static MBSS.

Definition 3.1.1. *Static mini-batch sub-sampling is conducted when the mini-batch, $\mathcal{B}$, used to evaluate approximations $L(\boldsymbol{x})$ and $\boldsymbol{g}(\boldsymbol{x})$ remains constant for the minimum duration of conducting a line search along a descent direction $\boldsymbol{d}_n$. Consider iteration, n, of an optimizer, given descent direction $\boldsymbol{d}_n$, the mini-batch $\mathcal{B}_n$ is sampled and kept constant when computing:*

$$\bar{L}(\boldsymbol{x}) = \frac{1}{|\mathcal{B}_n|}\sum_{b\in\mathcal{B}_n}\ell(\boldsymbol{x};\ \boldsymbol{t}_b), \tag{3.5}$$

and approximate gradient

$$\bar{\boldsymbol{g}}(\boldsymbol{x}) = \frac{1}{|\mathcal{B}_n|}\sum_{b\in\mathcal{B}_n}\nabla\ell(\boldsymbol{x};\ \boldsymbol{t}_b), \tag{3.6}$$

along $\boldsymbol{d}_n$. The mini-batch is only updated again for iteration $n+1$. The overhead bar is used to identify approximations evaluated using static mini-batch sub-sampling as $\bar{L}(\boldsymbol{x})$ and $\bar{\boldsymbol{g}}(\boldsymbol{x})$ respectively. For mini-batch size $|\mathcal{B}_n|$ and full batch size M, there are a total of $K = \binom{M}{|\mathcal{B}_n|}$ combinations from which to draw mini-batches.

Given the operation of differentiable norms on only smooth activation functions, a number of smooth loss and derivative function instances can result along the search direction depending on the sampled mini-batch at the beginning of the iteration as depicted in Figure 3.1. The differences between the smooth loss and derivative functions computed from the various mini-batches are a direct result of differences in sampling error between individual mini-batches.

Secondly, the *number* of samples drawn from the training data set influences the variance of the sampling error between batches. To not distract from the focus of this investigation, once a batch size, $|\mathcal{B}|$, is chosen for an optimization run, it remains fixed for the duration of the analysis. This deliberate choice does not diminish the contribution of other investigations that have pointed out the advantages of changing the batch size [Friedlander and Schmidt, 2011].

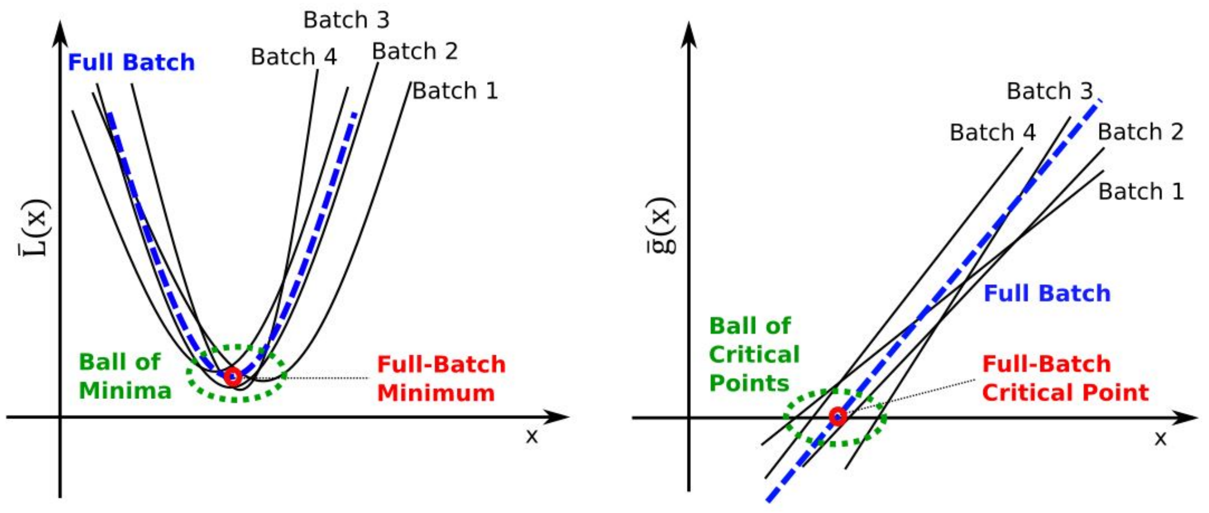


Figure 3.1: Illustration of the (a) full-batch loss function (blue) and four approximate loss functions computed using four static mini-batches along a descent direction $\boldsymbol{d}_n$. Similarly, (b) indicates the directional derivatives of these four static mini-batch sub-sampled loss functions along $\boldsymbol{d}_n$.

Lastly, the *way* the samples are drawn from the training data set influences the variance of the sampling error between batches. Usually, a mini-batch is drawn uniformly and independently from the training set [Chen et al., 2017]. Uniform sampling of mini-batches over the whole training set gives an expected loss $\mathbb{E}[L(\boldsymbol{x})] = \mathcal{L}(\boldsymbol{x})$ and expected gradient $\mathbb{E}[\boldsymbol{g}(\boldsymbol{x})] = \nabla\mathcal{L}(\boldsymbol{x})$, that is given by the full-batch computed loss and gradient functions [Tong and Liu, 2005]. It has been noted that MBSS may lead to the non-convergence of certain algorithms [Balles and Hennig, 2018], which can be improved by using stratified or active MBSS [Zhang et al., 2018]. In this investigation, all mini-batches are computed by drawing samples uniformly and independently from the training set.

Not only does MBSS benefit the computational cost of training, it may also aid convergence [Ruder, 2016], by allowing optimizers to overcome local minima in multi-modal loss functions [Choromanska et al., 2015, Dauphin et al., 2014, Goodfellow et al., 2015, Saxe et al., 2013]. This is based on the premise that local minima may vary between mini-batches. Though this premise is problem dependent, it seems to hold in general, when sampling using smaller batch sizes. In addition, descent directions computed from MBSS gradient functions produce a cone of possible descent directions distributed around the full-batch steepest descent direction, which may aid or delay convergence.

Notation: Explicit dependency on variables are occasionally omitted e.g. $\bar{L}$ instead of $\bar{L}(\boldsymbol{x})$. Sequences are denoted by subscripts e.g. $\bar{\boldsymbol{g}}_i$, denoting the gradient at the i^{th} iterate. Entry-wise products also known as the Hadamard or Schur products [Davis, 1962] are explicitly denoted using $\odot$. Directional derivative denotes a derivative or gradient vector projected along a direction. Gradient vectors are assumed to be column vectors and the vector transpose is indicated by superscript T. Note that the unit descent direction, $\boldsymbol{u}$, is distinct from the unscaled descent direction, $\boldsymbol{d}$, which are related by $\boldsymbol{u} = \frac{\boldsymbol{d}}{||\boldsymbol{d}||_2}$.

3.1.2 Optimization Formulations: Full-Batch Sampling

In general there are three formulations to define the solution or candidate solutions to an optimization problem using (3.1) and (3.2), i.e. computed using full-batch sampling, namely:

1. Formulation 1: Direct minimization of (3.1) [Arora, 2011, Floudas and Pardalos, 2009, Snyman and Wilke, 2018],

2. Formulation 2: Finding stationary points of (3.2), known as the optimality criterion [Arora,

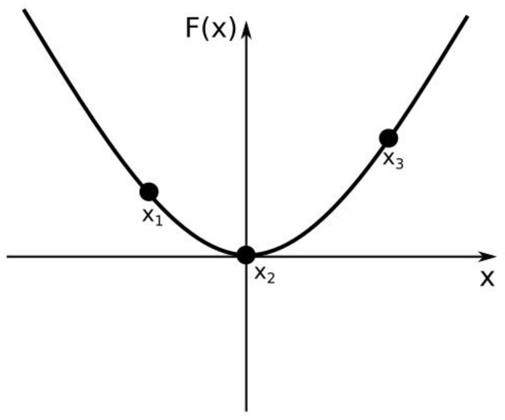
2011, Floudas and Pardalos, 2009, Snyman and Wilke, 2018], and

3. Formulation 3: Finding non-negative gradient projection points, $\boldsymbol{x}_{nngpp}$, [Snyman and Wilke, 2018, Wilke et al., 2013], as defined by the optimality condition in Definition 3.1.2.

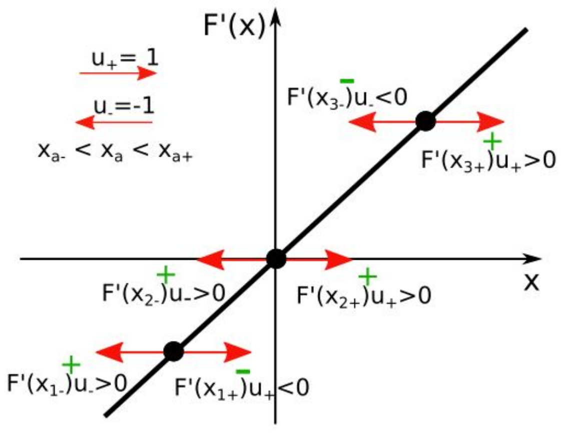
Definition 3.1.2. *Suppose that* $f : X \subseteq \mathbb{R}^p \rightarrow \mathbb{R}$ *is a real-valued function for which the* gradient $\nabla f(\boldsymbol{x})$ *is uniquely defined for every* $\boldsymbol{x} \in X$*. Then, a point* $\boldsymbol{x}_{nngpp} \in X$ *is a non-negative gradient projection point (NN-GPP) if there exists a real number* $r_u > 0$ *for every* $\boldsymbol{u} \in \{\boldsymbol{y} \in \mathbb{R}^p \;/\; \|\boldsymbol{y}\| = 1\}$ *such that*

$$\nabla f^T(\boldsymbol{x}_{nngpp} + \lambda \boldsymbol{u})\boldsymbol{u} \geq 0, \;\forall\; \lambda \in (0, r_u].$$

The NN-GPP refers to the directional derivative $\nabla f^{\mathrm{T}}(\boldsymbol{x}_{nngpp} + \lambda \boldsymbol{u})\boldsymbol{u}$ that is required to be non-negative for any direction $\boldsymbol{u}$ along a non-zero distance $\lambda \in (0, r_u]$. Note, that the gradient is evaluated at $\boldsymbol{x}_{nngpp} + \lambda \boldsymbol{u}$, i.e. away from $\boldsymbol{x}_{nngpp}$. Alternatively stated, NN-GPPs can be viewed as an optimality formulation that requires all directional derivatives of points in a non-empty ball, $B_\epsilon(\boldsymbol{x}) = \{\boldsymbol{x} \in \mathbb{R}^p : \|\boldsymbol{x} - \boldsymbol{x}_{nngpp}\|_2 < \mu,\; 0 < \mu < \infty,\; \boldsymbol{x} \neq \boldsymbol{x}_{nngpp}\}$, around the NN-GPP to be non-negative. The directional derivative, $\nabla f^{\mathrm{T}}(\boldsymbol{x})\boldsymbol{u}$, of a point in the ball $\boldsymbol{x} \in B_\epsilon$, is defined by the direction vector that connects the non-negative gradient projection point, $\boldsymbol{x}_{nngpp}$, to the point in the ball, $\boldsymbol{x}$, i.e. $\boldsymbol{u} = \frac{\boldsymbol{x} - \boldsymbol{x}_{nngpp}}{\|\boldsymbol{x} - \boldsymbol{x}_{nngpp}\|_2}$, which is then projected onto the gradient evaluated at $\boldsymbol{x}$.



(a) Univariate loss function, $F(x)$.



(b) Derivative, $F'(x)$.

Figure 3.2: (a) Univariate loss function and (b) derivative function. The non-negative gradient projection point (NN-GPP), x_{nngpp}, (Formulation 3) for the example univariate function coincides with $x_{nngpp} = x_2$, at which the derivative is also zero (Formulation 2) and the function value is a minimum (Formulation 1). NN-GPPs are defined using an optimality formulation that requires all directional derivatives of points in a non-empty ball, $B_\epsilon(x) = \{x \in \mathbb{R} : |x - x_{nngpp}| < \mu,\; 0 < \mu < \infty,\; x \neq x_{nngpp}\}$, around the NN-GPP to be non-negative.

Formulations 1-3 are well illustrated by the univariate loss function, $F(x)$, and derivative of the loss, $F'(x)$, depicted in Figure 3.2(a) and (b) respectively:

1. Formulation 1: x_2 is the minimum of $F(x)$.

2. Formulation 2: The only stationary point (candidate minimum) is x_2, since the necessary condition $F'(x_2) = 0$ holds. The candidate solution is an actual minimum if in addition to $F'(x_2) = 0$, $F''(x_2) > 0$ holds, which together forms the sufficiency conditions for x_2 to be a local minimum.

3. Formulation 3: Given, directions $u_- = -1$ and $u_+ = 1$, as well as $x_{a+} = x_a + \lambda$ and $x_{a-} = x_a - \lambda$ with $\lambda \in (0, r_u]$ for any point x_a; the only non-negative gradient projection point is x_2, as both directional derivatives, to the left, $F'(x_{2-})u_- > 0$, and to the right

$F'(x_{2+})u_+ > 0$ are both positive. This is in contrast to x_1 and x_3 that each have a negative directional derivative, respectively to the right and left, as indicated in Figure 3.2(b).

Consider the application of Formulations 1-3 to the actual loss function of a single hidden layer network applied to the Glass1 dataset [Prechelt, 1994], using the full training set in Figures 3.3(a) and (b). For given step length α along the steepest descent direction, $-\nabla\mathcal{L}(\boldsymbol{x}_0)$, Figure 3.3(a) depicts the full-batch univariate loss function

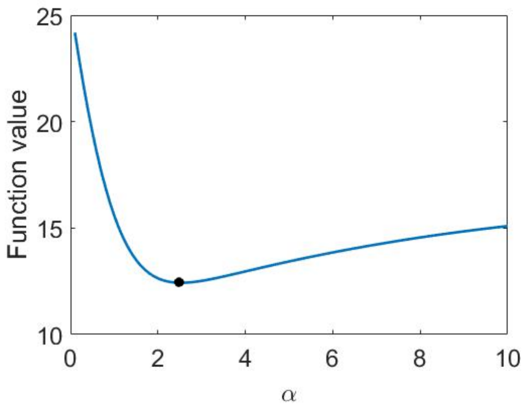
$$\mathcal{F}(\alpha) = \mathcal{L}(\boldsymbol{x}_0 - \alpha\nabla\mathcal{L}(\boldsymbol{x}_0)),$$

with Figure 3.3(b) depicting the corresponding directional derivative,

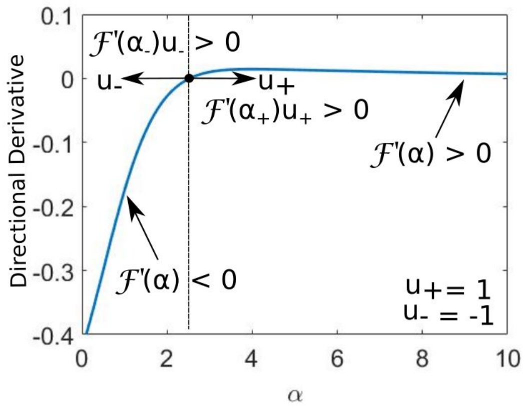
$$\mathcal{F}'(\alpha) = -\left(\nabla\mathcal{L}(\boldsymbol{x}_0 - \alpha\nabla\mathcal{L}(\boldsymbol{x}_0))\right)^{\mathrm{T}}\nabla\mathcal{L}(\boldsymbol{x}_0),$$

with each of the p components of $\boldsymbol{x}_0$ sampled from a uniform distribution over $[-0.1, 0.1]$. The minimum, stationary point and non-negative gradient projection point are equivalent and coincide with $\alpha = 2.5$.

Formulations 1-3 have been shown to be equivalent [Wilke et al., 2013], when restricting Equation (3.1) to the class of convex functions. Formulation 1 requires the direct minimization of Equation (3.1), while formulation 2 requires the roots of (3.2) to be computed. Formulation 3 is solved by finding sign changes, from negative to positive, in directional derivatives along descent directions until no more descent directions remain. Note sign changes from negative to positive along a descent direction implies a minimum without explicitly requiring the directional derivative to be zero at the minimum. In addition, a sign change from negative to positive implies a minimum based on derivative information. Hence, no additional second order condition is required to identify a minimum based on derivative information. For continuous and smooth convex functions, all three formulations are equivalent. However, as will become evident, these formulations distinguish themselves as the continuity and smoothness requirements for computed loss function are relaxed.



(a) Loss function.



(b) Directional derivative.

Figure 3.3: Typical loss function along the steepest descent direction in neural network training, with (a) function value and corresponding (b) directional derivative along the steepest descent direction as a function of step size α. The problem is constructed using the full training data of the Glass1 dataset [Prechelt, 1994] (for details see Section 3.3). The local minimum (Formulation 1) is indicated as a black dot in (a), while the stationary point (Formulation 2) and NN-GPP (Formulation 3) are indicated by a black dot in (b).

3.1.3 Optimization Formulations: Static Mini-Batch Sub-Sampling

As noted, static MBSS computes a sequence of approximate loss $\bar{L}(\boldsymbol{x})$ and gradient functions $\bar{\boldsymbol{g}}(\boldsymbol{x})$ to be considered by the optimizers for minimization. Since the mini-batch remains fixed along each search direction, the resulting loss and gradient functions are smooth, subject to the utilization of smooth activation functions and norm operators. Consequently, computing loss functions using static MBSS does not change the underlying smoothness and continuity of the computed loss function along a search direction. The boundedness of minimizers along a search direction may however be affected, i.e. different mini-batches may produce different minimizers along a fixed search direction, since the sampling error changes between mini-batches, resulting in a bias. In the limit case, it is possible that there is no minimizer when batch sizes are small, whereas the full-batch resolves a unique minimizer along the same search direction.

Consequently, Formulations 1-3 remain equivalent for smooth and continuous convex functions. However, the minimizers between different static MBSS loss functions are not unique. Instead, there is a set of solutions to which an optimizer will converge, each solution being the result of a particular mini-batch. This follows from considering Figures 3.1(a)-(b), which depicts four approximate loss functions, with four distinct minima. These four minima define a set of minima to which an optimizer can converge. For static MBSS, both the search direction and mini-batch is updated between iterations, resulting in a distribution of sampled loss functions that could be optimized at each iteration, where each sampled loss function has its own minimizer. The sampling error therefore introduces a biased minimizer estimate, which implies that convergence of an optimizer on static MBSS loss functions can only be guaranteed within some bounded distance, ϵ, that is $0 < \epsilon < \infty$, of the full-batch minimum. Although, static MBSS loss functions can be continuous, smooth and convex for a given mini-batch, the notion of convergence to a unique minimizer needs to be relaxed to a set of minimizers.

First, we need to ensure that our batches are sufficient to ensure that the smoothness, continuity and convexity characteristics of the full-batch function, $\mathcal{L}(\boldsymbol{x})$, are reflected in every independently sampled mini-batch loss function, $L(\boldsymbol{x})$:

Definition 3.1.3. *A mini-batch loss function, $L(\boldsymbol{x})$, and its gradient, $\boldsymbol{g}(\boldsymbol{x})$, are consistent, when the loss and gradient functions computed using the given mini-batch have the same smoothness, continuity and convexity characteristics as the full-batch loss function, $\mathcal{L}(\boldsymbol{x})$, and gradient, $\nabla\mathcal{L}(\boldsymbol{x})$.*

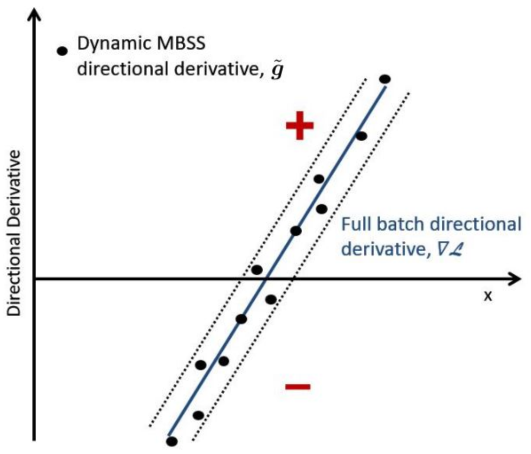
This avoids the computation of unbounded solutions along a search direction, which enables us to prove that minimization of static mini-batch sampled loss functions will converge to a finite ball of solutions, $B_\epsilon(\boldsymbol{x}) \subset \mathbb{R}^p$:

Theorem 3.1.1. *Consider a static mini-batch sub-sampled loss function $\bar{L}(\boldsymbol{x})$, of a continuous, smooth and convex full-batch loss function $\mathcal{L}(\boldsymbol{x})$, such that each computed mini-batch loss function is consistent. Then minimization of the static mini-batch sub-sampled loss function $\bar{L}(\boldsymbol{x})$ is guaranteed to converge to the ball $B_\epsilon(\boldsymbol{x}) = \{\boldsymbol{x} \in \mathbb{R}^p : \|\boldsymbol{x} - \boldsymbol{x}^*\|_2 < \epsilon,\ 0 < \epsilon < \infty,\ \boldsymbol{x} \neq \boldsymbol{x}^*\}$, where $\boldsymbol{x}^*$ is the minimum of full-batch loss function.*

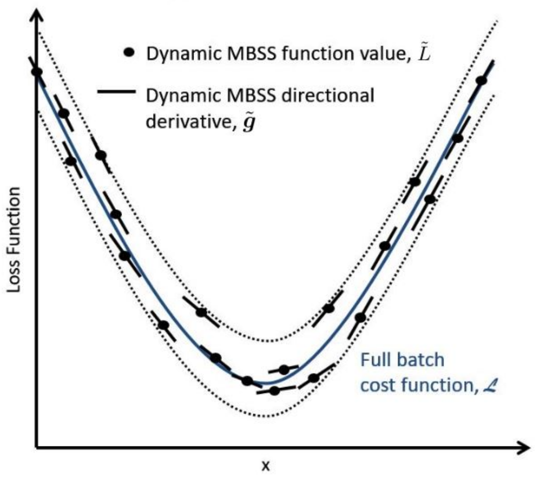
Proof. Given a training set of M samples, there are $K = \binom{M}{|\mathcal{B}|}$ ways to choose $|\mathcal{B}|$ samples from the training set. Since the mini-batch sub-sampled loss function is consistent, each of the K mini-batch sub-sampled loss functions $\bar{L}(\boldsymbol{x})$ is continuous, smooth and convex. Therefore, each of the K mini-batch sub-sampled loss functions $\bar{L}(\boldsymbol{x})$ has a unique minimizer given by $S = \{\boldsymbol{x}^{*1}, \boldsymbol{x}^{*2}, \ldots, \boldsymbol{x}^{*K}\}$. By selecting $0 < \epsilon < \infty$ such that $S \subset B_\epsilon(\boldsymbol{x})$, we construct $B_\epsilon(\boldsymbol{x}) = \{\boldsymbol{x} \in \mathbb{R}^p : \|\boldsymbol{x} - \boldsymbol{x}^*\|_2 < \epsilon,\ 0 < \epsilon < \infty,\ \boldsymbol{x} \neq \boldsymbol{x}^*\}$ to span all possible minimizers of the mini-batch sub-sampled loss functions $\bar{L}(\boldsymbol{x})$. Hence, minimization of the static mini-batch sub-sampled loss function $\bar{L}(\boldsymbol{x})$ implies minimization of one of the K mini-batch sub-sampled convex loss functions along the descent direction, i.e. a step of maximum improvement towards one of the minimizers in S along the descent direction. This in turn, is a guaranteed step of finite improvement towards $B_\epsilon(\boldsymbol{x})$ as $S \subset B_\epsilon(\boldsymbol{x})$, along the descent direction which proves the theorem. □

3.1.4 Optimization Formulations: Dynamic Mini-Batch Sub-Sampling

As an alternative to static MBSS, dynamic MBSS draws a new batch for every evaluation of the loss function:



(a) Loss function



(b) Directional derivative

Figure 3.4: Illustration of full-batch and dynamic mini-batch sub-sampled (MBSS) loss function and directional derivative along a unit descent direction, $\boldsymbol{u}$. For every evaluation of $L(\boldsymbol{x})$ and $\boldsymbol{g}(\boldsymbol{x})$ a new mini-batch is sampled. This introduces discontinuities in the loss and directional derivative functions, as every mini-batch, $\mathcal{B}$, has a different sampling error. This variance in sampling error is not present in the full-batch sampled loss function, which is smooth and continuous. Note that $\tilde{\boldsymbol{g}}(\boldsymbol{x})$ is analytically computed using back-propagation for a given mini-batch $\mathcal{B}$, which implies that $\tilde{L}(\boldsymbol{x})$ and $\tilde{\boldsymbol{g}}(\boldsymbol{x})$ are computed for the same mini-batch, $\mathcal{B}$.

Definition 3.1.4. *Dynamic mini-batch sub-sampling is conducted when the mini-batch, $\mathcal{B}$, used to evaluate approximations $L(\boldsymbol{x})$ and $\boldsymbol{g}(\boldsymbol{x})$ is updated for every evaluation of $L(\boldsymbol{x})$ and $\boldsymbol{g}(\boldsymbol{x})$ whilst conducting a line search along as descent direction $\boldsymbol{d}_n$. Therefore, given descent direction $\boldsymbol{d}_n$ at iteration n, requiring i loss function evaluations along the descent direction, then the mini-batch $\mathcal{B}_{n,i}$ is re-sampled for every loss function evaluation:*

$$\tilde{L}(\boldsymbol{x}) = \frac{1}{|\mathcal{B}_{n,i}|} \sum_{b \in \mathcal{B}_{n,i}} \ell(\boldsymbol{x};\ \boldsymbol{t}_b), \tag{3.7}$$

and approximate gradient

$$\tilde{\boldsymbol{g}}(\boldsymbol{x}) = \frac{1}{|\mathcal{B}_{n,i}|} \sum_{b \in \mathcal{B}_{n,i}} \nabla\ell(\boldsymbol{x};\ \boldsymbol{t}_b). \tag{3.8}$$

The overhead tilde is used to identify approximations evaluated using dynamic mini-batch sub-sampling as $\tilde{L}(\boldsymbol{x})$ and $\tilde{\boldsymbol{g}}(\boldsymbol{x})$ respectively. For mini-batch size $|\mathcal{B}_{n,i}|$ and full batch size M, there are a total of $K = \binom{M}{|\mathcal{B}_{n,i}|}$ combinations from which to draw mini-batches.

An explicit comparison between static and dynamic MBSS is presented in Section 3.4.6, which clearly demonstrates the computational efficiency of dynamic MBSS over static MBSS. However, dynamic MBSS significantly affects the characteristics of the computed loss and gradient functions, which are illustrated in Figure 3.4. Here, the loss function is sampled at discrete points along a given descent direction $\boldsymbol{d}_n$, while the mini-batch is updated or re-sampled at every evaluation of the loss function. The same batch to compute the loss function is used to compute the gradient, which in turn is projected onto the descent direction $\boldsymbol{d}_n$ to compute the depicted directional derivative. In addition to the dynamic MBSS loss and derivative functions, we also depict the full-batch loss and directional derivative functions. Firstly, it is important to note that both the dynamic MBSS loss and directional derivative functions are discontinuous. These discontinuities are a direct result of abrupt changes in the sampling error between mini-batches.

Given the utilization of only smooth activation functions operated on by differentiable norms, it follows that $\mathcal{L}(\boldsymbol{x})$ is everywhere differentiable. In turn, $\tilde{L}(\boldsymbol{x})$ in the classic sense is not differentiable everywhere as the limits of the derivative from left and right are not equal at discontinuities due to variance in the sampling error between mini-batches. However, we can compute the analytical sensitivity using back-propagation for any given $\boldsymbol{x}$ and mini-batch, $\mathcal{B}$. Following Wilke et al. [2013], we define the *associated derivative* for dynamic MBSS loss functions as follows:

Definition 3.1.5. *Let $f : X \subseteq \mathbb{R} \rightarrow \mathbb{R}$ be a real univariate dynamic mini-batch sub-sampled loss function, that is an everywhere defined point-wise discontinuous function. Given the mini-batch, $\mathcal{B}$, selected to evaluate the loss function at x, the* associated derivative *$f'^{a}(x)$ at x is then given by the left and right-hand derivative as x is approached when keeping $\mathcal{B}$ fixed. This is equivalent to the computed analytical sensitivity at x using mini-batch $\mathcal{B}$, which is easily computed using back-propagation.*

Secondly, the *associated gradient* for dynamic MBSS loss functions is given by:

Definition 3.1.6. *Let $f : X \subseteq \mathbb{R}^p \rightarrow \mathbb{R}$ be a dynamic mini-batch sub-sampled loss function that is an everywhere defined point-wise discontinuous function. The* associated gradient *$\nabla_a f(\boldsymbol{x})$ for $f(\boldsymbol{x})$ at $\boldsymbol{x}$ is defined as the vector of partial derivatives, where each partial derivative is an* associated derivative *as given by Definition 3.1.5.*

It follows from Definitions 3.1.5 and 3.1.6 that the *associated gradient* defines point-wise discontinuous dynamic MBSS loss functions to be differentiable everywhere. For compactness we will consider references to the gradient of a dynamic MBSS loss function, $\tilde{\boldsymbol{g}}(\boldsymbol{x})$, to imply the *associated gradient* of $\tilde{L}(\boldsymbol{x})$.

For the stochastic setting we extend the deterministic notion of a non-negative gradient projection point (NN-GPP) to incorporate the stochastic nature of the loss function due to dynamic MBSS as follows:

Definition 3.1.7. *Suppose that $f : X \subseteq \mathbb{R}^p \rightarrow \mathbb{R}$ is a real-valued function for which the* gradient *$\nabla f(\boldsymbol{x})$ is uniquely defined for every $\boldsymbol{x} \in X$. Then, a point $\boldsymbol{x}_{snngpp} \in X$ is a stochastic non-negative gradient projection point (SNN-GPP) if there exists a real number $r_u > 0$ for every $\boldsymbol{u} \in \{\boldsymbol{y} \in \mathbb{R}^p \ / \ \|\boldsymbol{y}\| = 1\}$ such that*

$$\nabla f^T(\boldsymbol{x}_{snngpp} + \lambda \boldsymbol{u})\boldsymbol{u} \geq 0, \ \forall \ \lambda \in (0, r_u],$$

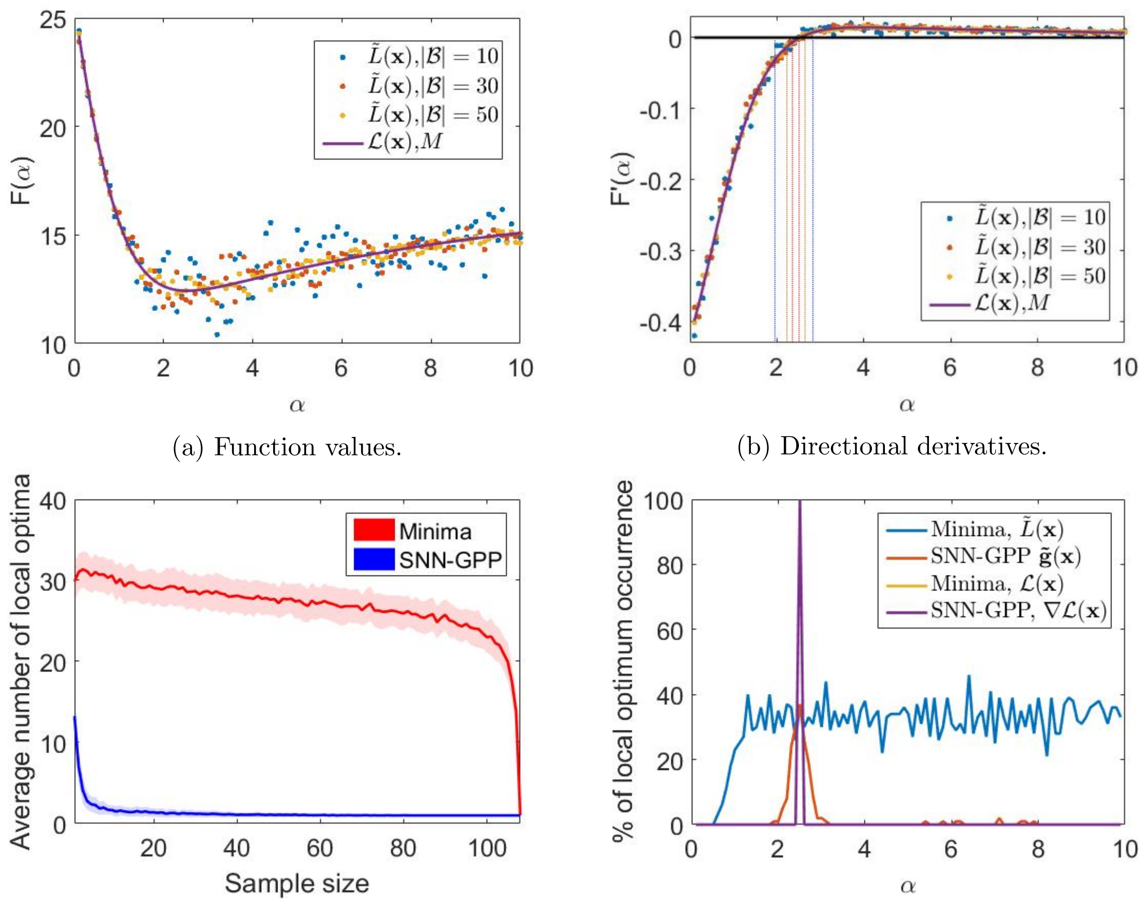
with a non-zero probability.

It follows that every NN-GPP is also a SNN-GPP, as it satisfies the above definition with a probability of 1.

Theorem 3.1.2. *Consider a dynamic mini-batch sub-sampled loss function $\tilde{L}(\boldsymbol{x})$, of a continuous, smooth and convex full-batch loss function $\mathcal{L}(\boldsymbol{x})$, such that each sampled mini-batch with associated $L(\boldsymbol{x})$ that is used to evaluate $\tilde{L}(\boldsymbol{x})$ is consistent. Then there exists a ball, $B_\epsilon(\boldsymbol{x}) = \{\boldsymbol{x} \in \mathbb{R}^p : \|\boldsymbol{x} - \boldsymbol{x}^*\|_2 < \epsilon, \ 0 < \epsilon < \infty, \ \boldsymbol{x} \neq \boldsymbol{x}^*\}$, that contains all the stochastic non-negative gradient projection points (SNN-GPPs), where $\boldsymbol{x}^*$ is the minimum of the full-batch loss function.*

Proof. Given a training set of M samples, then there are $K = \binom{M}{|\mathcal{B}|}$ ways to choose $|\mathcal{B}|$ samples from the training set. Since the mini-batch sub-sampled loss function is consistent, each of the K mini-batch sub-sampled loss functions $L(\boldsymbol{x})$ is continuous, smooth and convex. Therefore, each of the K mini-batch sub-sampled loss functions, $L(\boldsymbol{x})$, has a unique minimizer, which is also a non-negative gradient projection point (NN-GPP) or equivalently stochastic non-negative gradient projection point (SNN-GPP). The set of NN-GPP is given by $S = \{\boldsymbol{x}^{*1}, \boldsymbol{x}^{*2}, \ldots, \boldsymbol{x}^{*K}\}$. By selecting $0 < \epsilon < \infty$ such that $S \subset B_\epsilon(\boldsymbol{x})$, we construct $B_\epsilon(\boldsymbol{x}) = \{\boldsymbol{x} \in \mathbb{R}^p : \|\boldsymbol{x} - \boldsymbol{x}^*\|_2 < \epsilon, \ 0 < \epsilon < \infty, \ \boldsymbol{x} \neq \boldsymbol{x}^*\}$ to span all possible NN-GPPs of the mini-batch sub-sampled loss functions $L(\boldsymbol{x})$. This $B_\epsilon(\boldsymbol{x})$ then spans the set of SNN-GPPs, as SNN-GPPs are merely the consequence of stochastically selecting loss functions with their NN-GPP confined to S. It therefore follows that $B_\epsilon(\boldsymbol{x})$ spans all possible SNN-GPPs of the dynamic mini-batch sub-sampled loss function $\tilde{L}(\boldsymbol{x})$, which proves the theorem. □

To illustrate these definitions and visualize the ball of possible solutions along a search direction, reconsider the Glass1 dataset classification problem [Prechelt, 1994]. We now compute the dynamic MBSS loss function $\tilde{L}(\boldsymbol{x})$. Figures 3.5(a) and (b) show the function values and directional derivatives along the steepest descent direction for different mini-batch sizes $|\mathcal{B}| \in [10, 30, 50]$, which is compared to the full-batch loss function, i.e. $M = 108$. Consequently, a batch size of 10 result in $\approx 3.8 \times 10^{13}$ combinations, $|\mathcal{B}| = 30$ in $\approx 4.4 \times 10^{26}$ and $|\mathcal{B}| = 50$ in $\approx 1.8 \times 10^{31}$ combinations from which to draw a mini-batch.



(a) Function values.

(b) Directional derivatives.

(c) Minima versus mini-batch size.

(d) % local min. along search direction.

Figure 3.5: (a) Loss function and (b) directional derivative along the steepest descent direction with step size α for different mini-batch sizes, $|\mathcal{B}| \in [10, 30, 50]$, stochastically sub-sampled. The spatial variance in SNN-GPPs along the descent direction is indicated by vertical lines for different batch sizes. (c) The number of local minima and SNN-GPP that manifest for mini-batches sizes between $|\mathcal{B}| = 1$ and $M = 108$ sampled over the domain $0 \leq \alpha \leq 10$ divided into 101 increments. The quality of the solutions captured as the (d) variance in location of local minima and SNN-GPP over the domain $0 \leq \alpha \leq 10$ for a batch size of 10. The full-batch solution is at $\alpha^* = 2.5$.

Differences between Formulations 1-3 are evident when aiming to optimize dynamic mini-match sampled loss functions. Considering Formulation 1, it is evident that finding a minimum of $\tilde{L}(\boldsymbol{x})$ is both problematic and not representative of the full-batch loss function. Similarly, considering Formulation 2, requires the directional derivative to be zero, which (though numerically close to zero in some instances) may not exist for $\tilde{\boldsymbol{g}}(\boldsymbol{x})$. In fact, directional derivatives close to zero may lead to poor approximations of the true solution of $\nabla\mathcal{L}(\boldsymbol{x})$, as is evident with the diminishing directional derivatives for larger α. Finally, SNN-GPPs are significantly more robust approximations to the full-batch minimum $\boldsymbol{x}^*$, since they are localized more tightly in a ball around the full-batch NN-GPP. Evidently, for this problem larger batch sizes resolve tighter balls that confine the SNN-GPP as indicated by the vertical dotted lines in Figure 3.5(b), for the corresponding mini-batch sizes.

Given the context of line searches, we quantify the number and quality of local minima and SNN-GPPs along $\boldsymbol{d}_0 = -\nabla \mathcal{L}(\boldsymbol{x}_0)$ at some fixed initial point $\boldsymbol{x}_0$. In particular, we count the average number of local minima and SNN-GPPs over 100 reconstructions, i.e. $n = 1, \dots, 100$, of a line search $F(\alpha_{n,i}) = \tilde{L}(\boldsymbol{x}_0 + \alpha_{n,i}\boldsymbol{d}_0)$ for $\alpha_{n,i} = 0.1i,\ i = 1, \dots 100$, where the function value and gradient evaluations at each increment, i, are computed using a newly sampled mini-batch. The starting point, $\boldsymbol{x}_0$, is only sampled once at $n = 1$, from a uniform distribution between $[-0.1, 0.1]$ for each component of $\boldsymbol{x}_0$, and kept constant for $n = 2, \dots, 100$, resulting in the univariate function depicted in Figure 3.5(a) and (b).

A point, $\alpha_{n,i}$, is identified as a local minimum should $F(\alpha_{n,i-1}) > F(\alpha_{n,i}) < F(\alpha_{n,i+1})$, while $\alpha_{n,i}$ is deemed a SNN-GPPs when $F'(\alpha_{n,i-1}) < 0$ and $F'(\alpha_{n,i}) > 0$. Since there are 100 steps along the descent direction, the maximum number of local minima and SNN-GPP that can be found is 49 and 50 respectively. Conversely, if the function is monotonically decreasing, no local minima or SNN-GPP will be observed. We count the number of local minima and SNN-GPP over 100 constructed line searches, such as the line search shown in Figure 3.5(a) and (b), for batch sizes, $\mathcal{B} = \{1, 2, \dots, 108\}$. This allows us to quantify the effect of batch size on the average sensitivity of local minima and SNN-GPP. The full-batch function has only one local minimum or equivalently SNN-GPP within the depicted domain along the descent direction. The mean number of local minima and SNN-GPP are depicted in Figure 3.5(c), with shading denoting the standard deviation. As expected, it is evident that there is a unique local minimum and SNN-GPP when the full-batch is considered. As the batch size is reduced, the SNN-GPPs are still able to identify a unique solution, whereas the number of local minima grow rapidly. Even when the batch size is one, the number of SNN-GPPs that manifest is less than a third of the local minima.

To quantify the quality of the solutions, we depict in Figure 3.5(d), the location of the local minima and SNN-GPPs along the search direction for a batch size of 10. The full-batch optimum is located at $\alpha = 2.5$, around which the SNN-GPP are tightly clustered and mainly spread between $2 \leq \alpha \leq 3$, whereas the local minima are uniformly distributed between $1 \leq \alpha \leq 10$. Firstly, it is evident, that SNN-GPPs approximate the full-batch solution better than local minima when only optimizing the $\tilde{L}(\boldsymbol{x})$. Secondly, it is clear that convergence can only be guaranteed to a ball of solutions when dynamic MBSS is considered.

The implication is that line searches that aim to find SNN-GPPs should be more resilient and robust in approximating the location of true optima than line searches that aim to minimize along a descent direction, which may constantly be hindered by spurious local minima. The probability of SNN-GPPs is high around the full-batch optimum and diminishes as we move away from the full-batch optimum until the probability is a definite zero outside ball, B_ϵ. This has implications primarily for exact line searches, where we can only expect to resolve SNN-GPPs to within ball, B_ϵ, i.e. ϵ-accuracy for a given batch size. Usually, the larger the batch size the tighter ϵ that confines all SNN-GPPs.

3.1.5 Related Work

Supervised machine learning is often divided into "noisy" stochastic optimization problems [Byrd et al., 2012] related to smaller batch sizes (typically a single data point) and batch averaged approximation regimes that utilize larger batch sizes. As demonstrated in this chapter, both problems are discontinuous, with the size of the discontinuities decreasing as the batch size increases. Additional approaches to reduce the impact of discontinuities include dynamic sample sizes i.e. increasing the sample size as the optimizer converges [Byrd et al., 2012, Friedlander and Schmidt, 2011]. This results in continuous functions in the limit of the full-batch, but is not well suited for memory constrained datasets. Active sub-sampling [Zhang et al., 2018] is another approach to reduce the discontinuity size, in which the training data is split into separate datasets based on their error and then sub-sampled stochastically. Active sub-sampling [Zhang et al., 2018] has been shown to address convergence issues [Balles and Hennig, 2018] of well known machine learning optimizers such as Adam [Kingma and Ba, 2015]. The generalization of the optimization problem to find SNN-GPPs allows for a framework that is well suited for stochastic, dynamic and active sub-sampling. Global optimization strategies such as particle swarm optimization [Engelbrecht, 2005], genetic algorithms [Montana and Davis, 1989] and Bayesian combined genetic programming approaches [Marwala, 2007], are useful in the context of highly non-linear and multi-modal problems while only relying on loss function evaluations. However, these methods are well-known to be computationally demanding.

Stochastic gradient algorithms were introduced by Robbins-Monro [Robbins and Monro, 1951], and include developments such as Nesterov's dual averaging method [Nesterov, 2009]. Sub-gradient methods introduced by Shor [Shor, 1985a,b] are closely related to stochastic gradient algorithms. In fact, stochastic gradient descent (SGD) is a classical sub-gradient method i.e. steepest descent with a fixed learning rate [Bertsekas and Massachusetts Institute of Technology, 2015]. In turn, sub-gradient methods are closely related to the newly coined proximal-gradient methods [Bertsekas and Massachusetts Institute of Technology, 2015]. All these methods make use of *a priori* selected step sizes that are either constant or follow some schedule with known convergence characteristics. These learning rate parameters remain the most sensitive for sensible performance [Bergstra and Bengio, 2012], and are currently determined either by the user on a "trial and error" basis, or by computationally expensive automated means [Bergstra et al., 2011, Bergstra and Bengio, 2012, Jaderberg et al., 2017, Snoek et al., 2012]. An attempt to incorporate an adaptive learning rate using an inexact line search strategy based on the Wolfe conditions requires dynamic sub-sampling that iteratively increases the batch size up to the full-batch size [Byrd et al., 2012], which finally ensures a continuous problem free from discontinuities with well-known convergence characteristics. Another approach has been to conduct probabilistic line searches in a Bayesian optimization framework [Mahsereci and Hennig, 2017]. All the approaches discussed so far are referred to as function minimizers, i.e. they aim to find the minimum function value of the loss function.

Dynamic MBSS has seen limited application in line searches as these perform poorly or do not converge since the underlying assumptions on which the line searches were developed do not apply for stochastic sub-sampling. Theoretical developments of convergence proofs and estimating theoretical convergence rates include the well-known linear convergence rate of gradient descent methods for strongly-convex loss functions. Sub-linear convergence rates are achieved when f is only convex [Karimi et al., 2016]. A number of alternatives to strong convexity have been presented that include error bounds [Luo and Tseng, 1993], essential strong convexity [Liu et al., 2015], weak strong convexity [Gong and Ye, 2014], restricted Secant inequality [Zhang and Yin, 2013], quadratic growth with the Mangasarian-Fromovitz constraint qualification [Anitescu, 2000], Polyak-Lojasiewicz condition [Karimi et al., 2016] and associated derivative descent sequences [Wilke et al., 2013]. In line with the discontinuous nature of the stochastic loss function, associated derivative descent sequences are not based on assumptions of Lipschitz continuity or convexity but only assume associated derivative unimodal functions with convergent subsequences.

An alternative to function minimizers are gradient-only optimization strategies that solve for NN-GPPs as defined by the gradient-only optimization problem [Snyman and Wilke, 2018, Wilke et al., 2013]. NN-GPPs were specifically proposed to define solutions for discontinuous functions. In this chapter, we extend the notion of NN-GPP to stochastic NN-GPP (SNN-GPP) in more detail, and consider line searches specifically developed to find SNN-GPPs. We base our convergence proofs on associated derivative descent sequences.

3.1.6 Contribution

The gradient-only optimization problem which solves NN-GPPs [Snyman and Wilke, 2018, Wilke et al., 2013] is a generalized problem, which applies to full-batch training (the conventional minimization problem) and deterministic discontinuous problems. In this chapter we introduce the Stochastic NN-GPP (SNN-GPP) as applicable to training with dynamic MBSS (the discontinuous optimization problem). In practice NN-GPPs have been resolved using gradient-only line search (GOLS) strategies [Snyman and Wilke, 2018, Wilke et al., 2013]. In this chapter we adopt these principles to find SNN-GPPs in sub-sampled neural network loss functions and thereby automatically and adaptively determine the learning rates during supervised mini-batch training. We demonstrate its effectiveness and generality by comparing GOLS to minimization line search strategies over a large number of neural network problems. Once the capabilities of GOLS are established, we have the tools to optimize both continuous and discontinuous loss functions. Therefore, for the first time, we have the ability to directly compare static and dynamic MBSS in the context of optimization.

3.2 Method

As the emphasis of this chapter is on resolving step sizes and not the implications of search directions, we restrict ourselves to the mini-batch stochastic gradient descent (SGD) algorithm. We consider exact and inexact line searches that exclusively rely on loss function values [Arora, 2011, Floudas and Pardalos, 2009, Snyman and Wilke, 2018] or gradients [Snyman and Wilke, 2018, Wilke et al., 2013]. Additional benefits for considering line searches to resolve step sizes include the potential to consider higher order algorithms such as conjugate gradient and Quasi-Newton approaches to resolve curvature information of an underlying loss function [Arora, 2011, Floudas and Pardalos, 2009, Snyman and Wilke, 2018].

3.2.1 Stochastic Gradient Descent with Line Search Strategies

Consider stochastic gradient descent (SGD) algorithm modified to incorporate line searches, LS-SGD, as outlined in Algorithm 1. We denote α_{n,I_n} to be the step size for a given iteration, n. Line searches resolve α_{n,I_n} along an unscaled descent direction $\boldsymbol{d}_n$ from a starting point $\boldsymbol{x}_n$. This may require numerous function evaluations, i, to achieve. Therefore we denote $\alpha_{n,i}$ to be the step size at the i^{th} function evaluations of iteration, n. At termination α_{n,I_n} indicates the final step size, with I_n the number of function evaluations to resolve the step size for iteration, n.

Algorithm 1: LS-SGD: Stochastic gradient descent with line search

```
1 Select starting point, $\boldsymbol{x}_0$, and set $n = 0$.
2 Evaluate $\tilde{L}(\boldsymbol{x}_n)$ and $\boldsymbol{d}_n = -\tilde{\boldsymbol{g}}(\boldsymbol{x}_n)$.
3 Resolve step length, $\alpha_{n,I_n}$, using line search
4 Define $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_{n,I_n}\boldsymbol{d}_n$.
5 $n := n + 1$,
6 Continue when stop criterion and limit on number of iterations have not been met else
    stop.
7 Repeat steps 2 to 7.
```

Recall from Section 3.1, that F represented a univariate full-batch sampled loss function along a descent direction. In this section, we distinguish between static and dynamic MBSS representations of F following Definition 3.1.1 and 3.1.4, resulting in static MBSS sampled $\bar{F}$ and dynamic MBSS sampled $\tilde{F}$ respectively.

Subsequently, obtaining α_{n,I_n} for LS-SGD requires determining α for the univariate function

$$\tilde{F}_n(\alpha) = \tilde{L}(\boldsymbol{x}(\alpha)) = \tilde{L}(\boldsymbol{x}_n + \alpha\boldsymbol{d}_n), \tag{3.9}$$

with associated directional derivative along the descent direction $\boldsymbol{d}_n$,

$$\tilde{F}_n'(\alpha) = \frac{dF_n(\alpha)}{d\alpha} = \tilde{\boldsymbol{g}}^{\mathrm{T}}(\boldsymbol{x}(\alpha))\frac{d\boldsymbol{x}(\alpha)}{d\alpha} = \tilde{\boldsymbol{g}}^{\mathrm{T}}(\boldsymbol{x}_n + \alpha \cdot \boldsymbol{d}_n)\boldsymbol{d}_n. \tag{3.10}$$

Typically line searches are minimizers, i.e. find

$$\arg\min_{\alpha\in\mathbb{R}^+} \tilde{F}_n(\alpha), \tag{3.11}$$

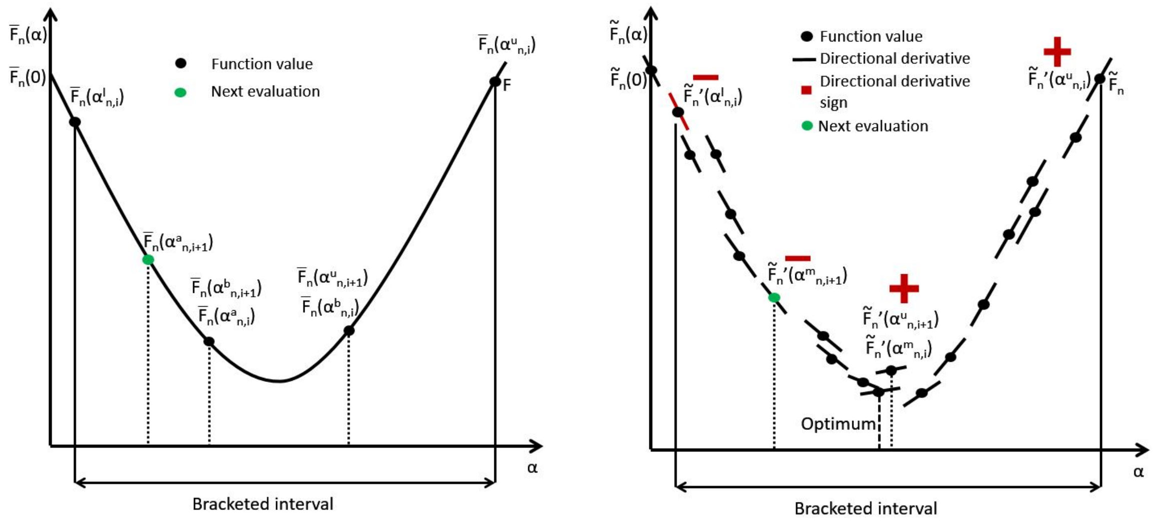
which defines functional value based line searches. Alternatively, line searches can be used to find SNN-GPPs i.e.

$$\arg \alpha_{snngpp} := \Big\{\alpha \in \mathbb{R} : \tilde{F}_n'(\alpha + \Delta\alpha)\Delta\alpha \geq 0 \Big| \, \forall \, |\Delta\alpha| < r | r > 0\Big\}, \tag{3.12}$$

to form the class of gradient-only line search strategies. For both approaches the desired points can be resolved within a small tolerance to give an exact line search, or approximately, which results in inexact line searches.

3.2.2 Exact line searches: Function Value based Golden Section (GS) and Gradient-only Line Search with Bisection (GOLS-B)

Function value based line searches assume twice differentiable smooth loss functions with a single minimizer isolated during the bracketing phase. We therefore present function value based line searches within the context of static MBSS loss functions, $\bar{F}$. Conversely, gradient-only line searches were developed for discontinuous loss functions. We therefore present gradient-only line searches within the context of dynamic MBSS loss functions, $\tilde{F}$, but remind the reader that gradient-only line searches is also applicable for smooth loss functions such as $\bar{F}$. We therefore distinguish between static and dynamic MBSS loss functions to highlight the differences between minimization and gradient-only line searches, but note that both minimization and gradient-only line searches can be applied to either static or dynamic MBSS loss functions, albeit with varying success. Exact line searches were initially developed to isolate minimizers for deterministic smooth functions, or NN-GPPs for deterministic discontinuous functions. Here, we apply exact line searches to converge to any stochastic minimizer or SNN-GPP bracketed. This is done to highlight the inherent quality of local minimizers vs SNN-GPPs for dynamic MBSS loss functions. For dynamic MBSS loss functions, the accuracy of the function value and gradient-only interval reduction schemes depends entirely on how representative the sampled loss function and gradient information is of the full-batch loss function, using the different optimality formulations (Section 3.1.4.).



$\bar{F}_n(\alpha)$
$\bar{F}_n(0)$
$\bar{F}_n(\alpha^l_{n,i})$
● Function value
● Next evaluation
$\bar{F}_n(\alpha^u_{n,i})$
F
$\bar{F}_n(\alpha^a_{n,i+1})$
$\bar{F}_n(\alpha^b_{n,i+1})$
$\bar{F}_n(\alpha^a_{n,i})$
$\bar{F}_n(\alpha^u_{n,i+1})$
$\bar{F}_n(\alpha^b_{n,i})$
α
Bracketed interval
$\tilde{F}_n(\alpha)$
$\tilde{F}_n(0)$
Function value
Directional derivative
Directional derivative sign
Next evaluation
$\tilde{F}_n'(\alpha^l_{n,i})$
$\tilde{F}_n'(\alpha^u_{n,i})$
$\tilde{F}_n$
$\tilde{F}_n'(\alpha^m_{n,i+1})$
$\tilde{F}_n'(\alpha^u_{n,i+1})$
$\tilde{F}_n'(\alpha^m_{n,i})$
Optimum
α
Bracketed interval

(a) Golden Section (b) Gradient-only Line Search with Bisection

Figure 3.6: Comparison between exact line searches that (a) minimize, such as the Golden Section (GS) method, versus (b) identify SNN-GPP by isolating sign changes from negative to positive using Gradient-Only Line Search with Bisection (GOLS-B).

Exact line searches first bracket a candidate solution and then refine the interval to find the minimum or SNN-GPP [Arora, 2011]. Refinement of a minimum requires more computation than isolating a SNN-GPP as illustrated in Figures 3.6(a) and (b). Four points forming three intervals are required to isolate a local minimum [Arora, 2011, Floudas and Pardalos, 2009, Snyman and

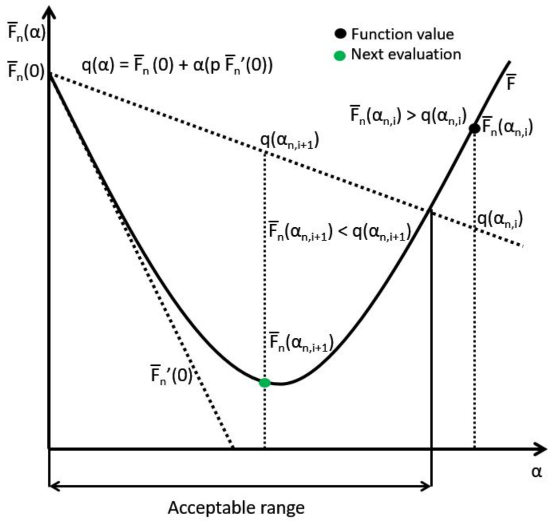
Wilke, 2018]). The optimal interval reduction is achieved using Golden Section (GS), that reduces the interval by 38% per iteration [Arora, 2011, Floudas and Pardalos, 2009, Snyman and Wilke, 2018]. Isolating a SNN-GPP is equivalent to isolating the directional derivative sign change from negative to positive along the descent direction, which can be done using bisection, i.e. three points forming two intervals, that reduces the interval by 50% per iteration [Snyman and Wilke, 2018]. We refer to this line search as gradient-only line search with bisection (GOLS-B) with pseudo-code listed in Appendix A.3.1. It is important to note, that finding only a sign change from negative to positive in the directional derivative along the descent direction enhances the robustness of gradient-only approaches against sampling errors, since magnitude variations in the directional derivative are ignored unless they result in sign changes from negative to positive. For both GS and GOLS-B the step sizes towards either the minimizer or SNN-GPP were resolved to a tolerance of 10^{-12}. Once a required interval is bracketed, the interval is reduced sequentially, based on the information presented to within the specified tolerance. For GS, the interval reduction as outlined in [Arora, 2011] is used. For GOLS-B, the sign of the mid-point can only be negative or positive, with the interval updated accordingly. This is repeated until the specified tolerance is achieved, resulting in the convergence to a point within ball, B_ϵ.

3.2.3 Inexact line search: Function value based Armijo's rule (ARLS) and Gradient-Only Line Search that is Inexact (GOLS-I)

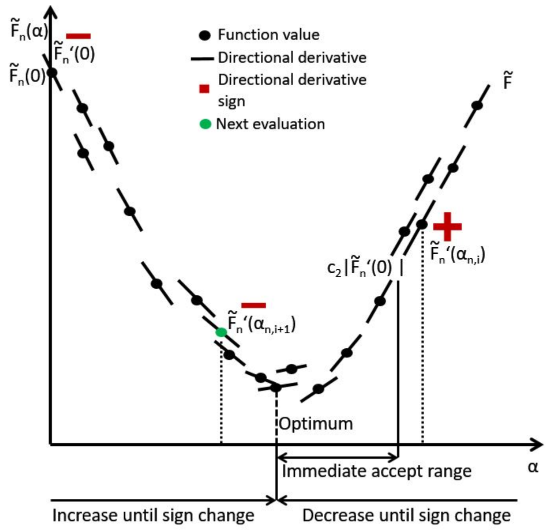
Inexact line searches define ranges for acceptable steps that are not:

1. too small by defining a lower bound for the steps, and
2. not too large by defining an upper bound for the steps.

Ensuring that step sizes are large enough is usually achieved by backtracking from a large step size until an acceptable step size has been found. Alternatively, a small step size is increased until an acceptable step size has been found. Inexact line searches balance accuracy with computational efficiency, and are therefore ideally suited for computationally competitive approaches to estimate step sizes.


$\bar{F}_n(\alpha)$
$\bar{F}_n(0)$
$q(\alpha) = \bar{F}_n(0) + \alpha(p\,\bar{F}_n'(0))$
● Function value
● Next evaluation
$\bar{F}$
$\bar{F}_n(\alpha_{n,i}) > q(\alpha_{n,i})$
$\bar{F}_n(\alpha_{n,i})$
$q(\alpha_{n,i+1})$
$q(\alpha_{n,i})$
$\bar{F}_n(\alpha_{n,i+1}) < q(\alpha_{n,i+1})$
$\bar{F}_n(\alpha_{n,i+1})$
$\bar{F}_n'(0)$
α
Acceptable range

(a) Function value inexact line search.



(b) gradient-only line search that is inexact.

Figure 3.7: Schematic diagrams for (a) the function value based inexact line search that is based on Armijo's rule (ARLS) and the (b) gradient-only line search that is inexact (GOLS-I) with the directional derivative slopes at the points of interest highlighted in red. Armijo's rule was developed for smooth functions, while the gradient-only inexact approach was developed for discontinuous functions.

The chosen function value based inexact line search, Armijo's rule line search (ARLS), is based on the practical and robust Armijo's Rule [Arora, 2011], that defines an upper bound to a domain of acceptable steps

$$\bar{F}_n(\alpha) < \bar{F}_n(0) + \alpha(p\bar{F}'_n(0)), \tag{3.13}$$

with $0 \leq p \leq 1$. For $p = 0$ any value below the level-set $F(0)$ is allowed, and for $p = 1$ the step-size for convex functions is reduced to 0. As a guideline $p = 0.2$ is preferred. In this investigation, a robust implementation of Armijo's Rule is realized that ensures the largest feasible step size is taken as the update step. If the initial step is acceptable the step size is increased until the condition is not satisfied and the largest acceptable step taken as the step. Should the initial step fail Armijo's rule, then the step size is reduced using backtracking until the first acceptable step size is found. This selective employment of advancement or backtracking ensures that the largest steps are taken. Step sizes are increased and decreased by a factor 2.

Our gradient-only line search that is inexact (GOLS-I), consists of two parts: An inexact sign change search method, and an immediate accept condition. The search for a sign change is conducted by considering the sign of the directional derivative at the initial guess, $(\frac{d\tilde{F}(\alpha_{n,0})}{d\alpha})$. If the sign is positive, the step size is reduced by a factor of $\eta = 2$, until the first negative directional derivative is observed. Conversely, should $(\frac{d\tilde{F}(\alpha_{n,0})}{d\alpha}) < 0$, the step size is grown by the same factor, η, until a positive directional derivative is obtained.

The magnitude of η determines a trade-off between computational cost and accuracy. Larger values of η make the line search more aggressive. However, this comes at the expense of accuracy, as SNN-GPPs may be over- or undershot considerably. Conversely, a small η value reduces the update increment size, increasing accuracy but also increasing the number of function evaluations per iteration needed to find SNN-GPPs. Although we found $\eta = 2$ to result in an acceptable balance between accuracy and computational cost, more formal sensitivity studies can be conduced in future work. However, for the purpose of comparing minimization and gradient-only line searches, we fix $\eta = 2$ for both inexact algorithms.

The immediate accept condition, if satisfied, results in the initial guess, $\alpha_{n,0}$ being chosen as the step size for the current iteration, n. This condition is given as follows:

$$0 < \frac{d\tilde{F}_n(\alpha_{n,0})}{d\alpha} \leq (c_2)|\frac{d\tilde{F}_n(0)}{d\alpha}|, \tag{3.14}$$

with $0 \leq c_2 \leq 1$. The notation of the "overshoot parameter", c_2, is intentionally chosen to allude to the strong Wolfe curvature condition [Arora, 2011]. However, note that the strong Wolfe condition accepts negative directional derivatives, whereas our condition only considers positive values. Here, $c_2 = 0$ implies that no overshoot is acceptable, no immediate accept condition is active and GOLS-I is reduced simply to a method that searches for a sign change. However, $c_2 = 1$ allows overshoots with positive directional derivatives of up to the same magnitude as the directional derivative at the current position, $|\frac{d\tilde{F}_n(0)}{d\alpha}|$. We kept the overshoot parameter constant at $c_2 = 0.9$, unless stated otherwise. The algorithmic details of this method are given in Appendix A.3.2.

As loss functions may have unbounded solutions for small batch sizes, we impose a maximum step size limit α_{max} on both ARLS and GOLS-I, in addition to a minimum step limit α_{min} given by

$$\alpha_{max} = \min(\frac{1}{\|\boldsymbol{d}_n\|_2}, 10^7), \tag{3.15}$$

$$\alpha_{min} = 10^{-8}. \tag{3.16}$$

The minimum step size truncates the number of function evaluations required per line search in the case where a spurious ascent direction is encountered. Here, line searches seek to drive $\alpha \rightarrow 0$, which can incur significant computational cost, when the step size is repeatedly divided by η. Conversely, the maximum step size restricts excessively large step sizes in the case of a uniformly descending search direction (akin to a linear function for e.g.), which occur when $|\mathcal{B}_{n,i}|$ is small. The contribution of $\frac{1}{\|\boldsymbol{d}_n\|_2}$ to the maximum step size ensures that small step size are conservatively enforced when the gradient norm is steep, while large updates are allowed when the gradient norm is moderate, which is often the case around saturated, stationary or saddle points.

Both inexact line search strategies require an initial step size. For the first iteration, this initial guess is set to $\alpha_{0,0} = \alpha_{min}$, to ensure that the step sizes are actively resolved by the line searches. In subsequent iterations the initial guess is set to be the resolved step size of the previous iteration, $\alpha_{n,0} = \alpha_{n-1,I_n}$.

3.2.4 Theoretical Basis

Before we present proofs of convergence of associated derivative descent sequences for associated derivative unimodal descendible functions, we present definitions for associated derivative convex descendible functions. This includes a number of piece-wise smooth step discontinuous functions, point-wise discontinuous functions and dynamic MBSS loss functions but excludes piece-wise linear continuous functions.

In building up to associated derivative convex descendible functions, we define the associated derivative convex descent sequence as follows:

Definition 3.2.1. *For a given sequence $\{\boldsymbol{x}^{\{k\}} \in X \subset \mathbb{R}^p : k \in \mathbb{P}\}$ suppose $\nabla_a f(\boldsymbol{x}^{\{k\}}) \neq \mathbf{0}$ for some k and $\boldsymbol{x}^{\{k\}} \notin B_\epsilon$ with B_ϵ defined in Theorem 3.1.2. Then the sequence $\{\boldsymbol{x}^{\{k\}}\}$ is an associated derivative convex descent sequence for $f : X \to \mathbb{R}$, if an associated sequence $\{\boldsymbol{u}^{\{k\}} \in \mathbb{R}^p : k \in \mathbb{P}\}$ may be generated such that if $\boldsymbol{u}^{\{k\}}$ is a descent direction from the set of all possible descent directions at $\boldsymbol{x}^{\{k\}}$, i.e. $\nabla_a f^T(\boldsymbol{x}^{\{k\}})\boldsymbol{u}^{\{k\}} < 0$ then*

$$\nabla_a f^T(\boldsymbol{x}^{\{k+1\}})\boldsymbol{u}^{\{k\}} < 0, \text{ for } \boldsymbol{x}^{\{k\}} \neq \boldsymbol{x}^{\{k+1\}}. \quad (3.17)$$

We can now define the class of associated derivative convex descendible multivariate functions

Definition 3.2.2. *A multivariate function $f : X \subseteq \mathbb{R}^p \to \mathbb{R}$ is associated derivative convex descendible if $\boldsymbol{x}^{\{0\}} \in \mathbb{R}^p$ and $\{\boldsymbol{x}^{\{k\}}\}$ is an associated derivative descent sequence, as defined in Definition 3.2.1, for f with initial point $\boldsymbol{x}^{\{0\}}$, then every subsequence of $\{\boldsymbol{x}^{\{k\}}\}$ converges. The limit of any convergent subsequence of $\{\boldsymbol{x}^{\{k\}}\}$ is $\{\boldsymbol{x}^{\{k\}}\} \in B_\epsilon$ as $k \to \infty$, with B_ϵ as defined in Theorem 3.1.2.*

Note. *Performing exact gradient-only line searches result in associated derivative descent sequences, whilst our proposed inexact line search may or may not depending on the chosen parameter values. Although we consider parameter values in this study for which strict associated derivative descent sequences do not hold (a degree of overshoot is allowed), the generated subsequences are convergent for the problems considered in this investigation. This is due to the implementation of our modified Wolfe curvature condition, which causes the norm of the gradient to decrease on average. However, the efficiency of this condition depends on the variance in the sampling error of the gradients.*

Theorem 3.2.1. *Let $f : X \subseteq \mathbb{R}^p \to \mathbb{R}$ be a associated derivative convex descendible multivariate function as defined by Definition 3.2.2. Let $\boldsymbol{x}^{\{0\}} \in \mathbb{R}^p$ and $\{\boldsymbol{x}^{\{k\}}\}$ be an associated derivative descent sequence, as defined in Definition 3.2.1, for f with initial point $\boldsymbol{x}^{\{0\}}$, then by definition every subsequence of $\{\boldsymbol{x}^{\{k\}}\}$ converges. The limit of any convergent subsequence of $\{\boldsymbol{x}^{\{k\}}\}$ is B_ϵ as defined in Theorem 3.1.2.*

Proof. It follows our assertion that f is a associated derivative convex descendible multivariate function given by Definition 3.2.2, that every derivative descent subsequence $\{\boldsymbol{x}^{\{k\}}\}$ converges.

Now let $\{\boldsymbol{x}^{\{k\}_m}\}$ be a convergent subsequence of $\{\boldsymbol{x}^{\{k\}}\}$ and let $\boldsymbol{x}^{m*}$ be its limit. Suppose, contrary to the second assertion of the theorem, that $\boldsymbol{x}^{m*} \notin B_\epsilon$ as defined in Theorem 3.1.2. Since we assume that $\boldsymbol{x}^{m*} \notin B_\epsilon$, and by Definition 3.2.1, there exists a $\boldsymbol{x}^{m*} + \lambda\boldsymbol{u}$ for $\lambda \neq 0 \in \mathbb{R}$ and $\boldsymbol{u} \in \{\boldsymbol{y} \in R^p : \|\boldsymbol{y}\|_2 = 1\}$ such that $\nabla_a f(\boldsymbol{x}^{m*} + \delta\boldsymbol{u})\boldsymbol{u} < 0$, which contradicts our assumption that $\boldsymbol{x}^{m*}$ is the limit of the subsequence $\{\boldsymbol{x}^{\{k\}_m}\}$. Therefore, for $\boldsymbol{x}^{m*}$ to be the limit of an associated derivative descent subsequence $\{\boldsymbol{x}^{\{k\}_m}\}$, $\boldsymbol{x}^{m*} \in B_\epsilon$ as defined in Theorem 3.1.2, which completes the proof. $\square$

The class of associated derivative convex multivariate functions can be generalized to include associated derivative unimodal multivariate functions.

Definition 3.2.3. *Consider a dynamic mini-batch sub-sampled loss function $\tilde{L}(\boldsymbol{x})$, of a continuous, smooth and unimodal full-batch loss function $\mathcal{L}(\boldsymbol{x})$, such that each computed mini-batch loss function is consistent. The class of associated derivative unimodal multivariate functions is defined when there exists a ball, $B_\epsilon(\boldsymbol{x}) = \{\boldsymbol{x} \in \mathbb{R}^p : \|\boldsymbol{x} - \boldsymbol{x}^*\|_2 < \epsilon,\ 0 < \epsilon < \infty,\ \boldsymbol{x} \neq \boldsymbol{x}^*\}$, that contains all the stochastic non-negative gradient projection points (SNN-GPPs), where $\boldsymbol{x}^*$ is the minimum of the full-batch loss function.*

We can now extend the notation of associated derivative convex descent sequences to associated derivative unimodal descent sequences:

Definition 3.2.4. *For a given sequence $\{\boldsymbol{x}^{\{k\}} \in X \subset \mathbb{R}^p : k \in \mathbb{P}\}$ suppose $\nabla_a f(\boldsymbol{x}^{\{k\}}) \neq \mathbf{0}$ for some k and $\boldsymbol{x}^{\{k\}} \notin B_\epsilon$ with B_ϵ defined in Definition 3.2.3. Then the sequence $\{\boldsymbol{x}^{\{k\}}\}$ is a associated derivative unimodal descent sequence for $f : X \to \mathbb{R}$, if an associated sequence $\{\boldsymbol{u}^{\{k\}} \in \mathbb{R}^p : k \in \mathbb{P}\}$ may be generated such that if $\boldsymbol{u}^{\{k\}}$ is a descent direction from the set of all possible descent directions at $\boldsymbol{x}^{\{k\}}$, i.e. $\nabla_a f^T(\boldsymbol{x}^{\{k\}})\boldsymbol{u}^{\{k\}} < 0$ then*

$$\nabla_a f^T(\boldsymbol{x}^{\{k+1\}})\boldsymbol{u}^{\{k\}} < 0,\ \textit{for}\ \boldsymbol{x}^{\{k\}} \neq \boldsymbol{x}^{\{k+1\}}. \tag{3.18}$$

This enables the definition of associated derivative unimodal descendible multivariate functions

Definition 3.2.5. *A multivariate function $f : X \subseteq \mathbb{R}^p \to \mathbb{R}$ is associated derivative unimodal descendible if $\boldsymbol{x}^{\{0\}} \in \mathbb{R}^p$ and $\{\boldsymbol{x}^{\{k\}}\}$ is an associated derivative descent sequence, as defined in Definition 3.2.4, for f with initial point $\boldsymbol{x}^{\{0\}}$, then every subsequence of $\{\boldsymbol{x}^{\{k\}}\}$ converges. The limit of any convergent subsequence of $\{\boldsymbol{x}^{\{k\}}\}$ is $\{\boldsymbol{x}^{\{k\}}\} \in B_\epsilon$ as $k \to \infty$, with B_ϵ as defined in Definition 3.2.3.*

This enables us to extend our class of associated derivative descendible convex multivariate functions to include associated derivative unimodal descendible multivariate functions.

Theorem 3.2.2. *Let $f : X \subseteq \mathbb{R}^p \to \mathbb{R}$ be a associated derivative unimodal descendible multivariate function as defined by Definition 3.2.5. Let $\boldsymbol{x}^{\{0\}} \in \mathbb{R}^p$ and $\{\boldsymbol{x}^{\{k\}}\}$ be an associated derivative descent sequence, as defined in Definition 3.2.4, for f with initial point $\boldsymbol{x}^{\{0\}}$, then by definition every subsequence of $\{\boldsymbol{x}^{\{k\}}\}$ converges. The limit of any convergent subsequence of $\{\boldsymbol{x}^{\{k\}}\}$ is $\{\boldsymbol{x}^{\{k\}}\} \in B_\epsilon$ as $k \to \infty$, with B_ϵ as defined in Definition 3.2.3.*

Proof. It follows our assertion that f is a associated derivative unimodal descendible multivariate function given by Definition 3.2.5, that every derivative descent subsequence $\{\boldsymbol{x}^{\{k\}}\}$ converges.

Now let $\{\boldsymbol{x}^{\{k\}_m}\}$ be a convergent subsequence of $\{\boldsymbol{x}^{\{k\}}\}$ and let $\boldsymbol{x}^{m*}$ be its limit. Suppose, contrary to the second assertion of the theorem, that $\boldsymbol{x}^{m*} \notin B_\epsilon$ as defined in Definition 3.2.3. Since we assume that $\boldsymbol{x}^{m*} \notin B_\epsilon$, and by Definition 3.2.4, there exists a $\boldsymbol{x}^{m*} + \lambda\boldsymbol{u}$ for $\lambda \neq 0 \in \mathbb{R}$ and $\boldsymbol{u} \in \{\boldsymbol{y} \in R^p : \|\boldsymbol{y}\|_2 = 1\}$ such that $\nabla_a f(\boldsymbol{x}^{m*} + \delta\boldsymbol{u})\boldsymbol{u} < 0$, which contradicts our assumption that $\boldsymbol{x}^{m*}$ is the limit of the subsequence $\{\boldsymbol{x}^{\{k\}_m}\}$. Therefore, for $\boldsymbol{x}^{m*}$ to be the limit of an associated derivative unimodal descent subsequence $\{\boldsymbol{x}^{\{k\}_m}\}$, $\boldsymbol{x}^{m*} \in B_\epsilon$ as defined in Definition 3.2.3, which completes the proof. □

We consider the latter class of associated derivative unimodal descendible multivariate functions as an appropriate definition for a large class of machine learning loss functions.

3.3 Numerical Studies

The primary aim of our numerical investigations is to demonstrate that GOLS presents a more effective means than function value based line searches to adaptively resolve step sizes for dynamic MBSS neural network loss functions. At each iteration, an appropriate step size has to be resolved along a new descent direction. All our analyses are therefore conducted for a fixed number of iterations, while the various line searches have distinct computational costs per iteration. We therefore present our results in terms of *function evaluations* to allow the performance and computational efficiency of the various line searches to be investigated. Since we compare methods that use function values to those that use gradients, we need to equate the computational cost of various computed information. In this chapter, one *gradient evaluation* equates to two *function evaluations*, since a function evaluation is generated by a forward pass of data through the network, while a gradient evaluation requires both a forward and a backward pass (via backpropagation) through the network. For analyses that focus on the quality with which the step size was resolved, we present results in terms of iterations as opposed to function evaluations.

First we conduct an extensive analysis on a diverse set of foundational classification problems to demonstrate that our GOLS formulations are effective over different datasets and architectures in the context of dynamic MBSS. We show that both GOLS-B and GOLS-I resolve step sizes adaptively, with regards to the characteristics of a specific problem. After establishing the validity of our GOLS formulations, we subsequently have the necessary means by which to directly compare static and dynamic MBSS in the context of line searches. This direct comparison using one method is enabled by the fact that the SNN-GPP, on which GOLS is based, is applicable to resolve step sizes for both static MBSS loss functions that are continuous, and dynamic MBSS loss functions that are discontinuous. To the best of our knowledge, this is the first such comparison, that shows the benefits of dynamic MBSS over static MBSS.

We then apply GOLS-I to a Variational Autoencoder (VAE) example and use this example to explore the effect of estimating SNN-GPPs inexactly, in particular within the context of LS-SGD search directions for poorly scaled loss functions. We show that under these circumstances, conservative line searches that isolate SNN-GPP locations are less competitive than deliberate overshoot of SNN-GPPs. By overshooting SNN-GPPs along the primary, steep-descent directions efficient progress along secondary, more gradual descent directions are achieved. This is in addition to potential benefits achieved when consecutive steepest descent directions are non-orthogonal. We the proceed to demonstrate that these performance gains are purely a result of improvements to ineffective steepest descent directions, as overshooting SNN-GPPs for LS-SGD can be outperformed by accurately resolving SNN-GPPs for the improved search directions of Adagrad. This investigation shows that GOLS is reliable in finding SNN-GPP for two classes of descent directions; and demonstrates the capability of GOLS to adaptively resolve step sizes in dynamic MBSS loss functions over different datasets, architectures, loss functions and search directions.

3.3.1 Diverse Foundational Classification Problems

In our numerical investigations we consider 22 classification problem datasets that cover from 150 to 70 000 observations per dataset. Our earliest dataset was made available in 1936 and the latest in 2016. The datasets vary in input features from 4 to 784 and classes from 2 to 29. This range in characteristics gives the selected problems a large variety of loss function landscapes for the different optimization algorithms to navigate. The primary aim of our numerical studies is to demonstrate, that the performance trends observed of GOLS are general across vastly different datasets when evaluating loss functions with dynamic MBSS. Details concerning the datasets are given in Table 3.1 and the corresponding neural network architectures implemented for the different datasets are given in Table 3.2. Fully connected neural network layers with one, as well as two hidden layers were considered for all of the datasets but one. For the two hidden layer case, each hidden layer was given the same number of nodes, as shown in Table 3.2. Resulting in a total of 43 classification problems to be solved.

We consider all four optimizers outlined, namely the minimizing line searches (GS and ARLS) and the gradient-only line searches (GOLS-B and GOLS-I). All optimizers completed 3000 iterations for an arbitrarily chosen constant mini-batch size of 10 over all datasets. The variance in sampling error of 10 samples on different datasets varies resulting in a range of sampling error

Dataset properties					
Dataset ref. no.	**Dataset name**	**Reference**	**Observations**	**Features**	**Classes**
1	Cancer1	Prechelt [1994]	699	9	2
2	Card1	Prechelt [1994]	690	51	2
3	Diabetes1	Prechelt [1994]	768	8	2
4	Gene1	Prechelt [1994]	3175	120	3
5	Glass1	Prechelt [1994]	214	9	6
6	Heartc1	Prechelt [1994]	920	35	2
7	Horse1	Prechelt [1994]	364	58	3
8	Mushroom1	Prechelt [1994]	8124	125	2
9	Soybean1	Prechelt [1994]	683	35	19
10	Thyroid1	Prechelt [1994]	7200	21	3
11	Abalone	Nash et al. [1994]	4177	8	29
12	Iris	Fisher [1936]	150	4	3
13	H.A.R.	Anguita et al. [2012]	10299	561	6
14	Bankrupted Co. (yr. 1)	Zięba et al. [2016]	7027	64	2
15	Defaulted Credit Cards	Yeh and Lien [2009]	30000	24	2
16	Forests	Johnson et al. [2012]	523	27	4
17	FT Clave	Vurkaç [2011]	10800	16	4
18	Sensor-less Drive	Paschke et al. [2013]	58509	48	11
19	Wilt	Johnson et al. [2013]	4839	5	2
20	Biodegradable compounds	Mansouri et al. [2013]	1054	41	2
21	Simulation failures	Lucas et al. [2013]	540	20	2
22	MNIST Handwriting	Lecun et al. [1998]	70000	784	10

Table 3.1: Properties of the considered datasets for the investigation.

ANN properties				
Dataset name	**Input nodes**	**Hidden layer nodes**	**Hidden layers**	**Output nodes**
Cancer1	9	8	1/2	2
Card1	51	5	1/2	2
Diabetes1	8	7	1/2	2
Gene1	120	9	1/2	3
Glass1	9	5	1/2	6
Heartc1	35	3	1/2	2
Horse1	58	2	1/2	3
Mushroom1	125	8	1/2	2
Soybean1	35	3	1/2	19
Thyroid1	21	8	1/2	3
Abalone	8	7	1/2	29
Iris	4	3	1/2	3
H.A.R.	561	7	1/2	6
Bankrupted Co. (yr. 1)	64	35	1/2	2
Defaulted Credit Cards	24	23	1/2	2
Forests	27	6	1/2	4
FT Clave	16	15	1/2	4
sensor-less Drive	48	47	1/2	11
Wilt	5	4	1/2	2
Biodegradable compounds	41	8	1/2	2
Simulation failures	20	8	1/2	2
MNIST	784	30	1	10

Table 3.2: Properties of the neural network architecture for each dataset.

variance across the datasets. Alternatively, the mini-batch size can be optimized to achieve comparable variance across the datasets [Radiuk, 2017], but this is considered to be outside of the scope of this investigation.

For all the given datasets, the data was split into training, validation and test datasets with a 2:1 ratio between number of samples in training to validation and training to test datasets. This three-way split enables the comparison of results between the validation and test datasets, which, if comparable, confirms that the test dataset is unbiased and has a sufficient number of samples. Performance was measured for training, validation and test datasets at every iteration using the mean squared error (MSE) loss function, expressed by Equation (A.3) in Appendix A.2. The optimization runs of 3000 iterations are repeated 10 times using exactly the same starting points between network architectures and optimizers, by drawing random values between $[-0.1, 0.1]$ for each neural network weight [Prechelt, 1994]. This small initialization range is deliberate to ensure that all initial Sigmoid activation functions are in their sensitive domain.

3.3.2 Variational Autoencoder Training

To investigate whether GOLS generalize to a different class of network architecture, we train a Variational Autoencoder (VAE) on the MNIST dataset using GOLS-I. A Pytorch 1.0 [pytorch.org, 2019] implementation of the VAE was sourced from Zou [Zuo and Chintala, 2018], which in turn implemented the architecture proposed by Kingma and Welling [2013]. It was subsequently modified to include and use GOLS-I and other variants using LS-SGD. Note, that the loss function in this example is a function of KL-divergence and the binary cross entropy, thus presenting GOLS-I with a new class of loss functions to navigate.

In this investigation we compare GOLS-I to three instances of LS-SGD with fixed step sizes of $\alpha_{n,I_n} = 10^{-5}$, $\alpha_{n,I_n} = 10^{-4}$ and $\alpha_{n,I_n} = 10^{-3}$. The constant learning rates were manually chosen and selected such that one learning rate is too high, one appropriate and one too low, each separated by an order of magnitude. We also explore variants of GOLS-I that allow overestimation of the step size (GOLS-Max), and others that seek to be more conservative with regards to finding the location of the sign change (GOLS-Back and modified GOLS-I with $c_2 = 0$). We do this to illustrate that: Firstly, when using various approaches to estimate univariate SNN-GPPs along descent directions, these approaches estimate similar locations of univariate SNN-GPPs. Secondly, that finding optima along descent directions alone is not necessarily conducive to effective training when using LS-SGD, as instead a large amount of overshoot can be beneficial. This occurs when the loss function has significantly varying curvature characteristics in different directions. In such cases the primary direction of improvement may not direct the algorithm to effective progress towards the high-dimensional SNN-GPP. Instead, secondary directions with lower curvature may lead to more effective updates. Consequently, by overstepping the univariate SNN-GPPs along the primary direction, implies a larger update towards the high-dimensional SNN-GPP affected by the secondary direction. The implication of this is that resolving sign changes along poor descent directions may lead to slower convergence, than non-optimal updates along the same directions. Line searches are therefore sensible, when locating univariate SNN-GPPs along primary directions implies significant progress towards the SNN-GPP of the high dimensional problem. We demonstrate this by significantly improving training of this problem with the use of GOLS-I to estimate SNN-GPP along Adagrad's [Duchi et al., 2011] descent directions.

3.3.3 Comparison between static and dynamic mini-batch sub-sampling

To conclude, we select a training problem proposed by Mahsereci and Hennig [2017], namely that of the NetII architecture with the CIFAR10 [Krizhevsky and Hinton, 2009] dataset, to compare the performance of static and dynamic MBSS within GOLS-I. NetII is a 3-hidden-layer classifier network with 1000, 500 and 250 hidden nodes respectively. The number of input nodes is 3072, as determined by the 32x32 resolution and three colour channels of the images in the CIFAR10 dataset. The images belong to 10 different classes, dictating the number of output nodes. Each image is pre-processed by applying the standard transform to every image channel. NetII implements Tanh activation functions, and is evaluated using the MSE loss. As prescribed by Mahsereci and Hennig [2017], training is conducted using using Batch1 of the CIFAR10 dataset, over 10,000 function evaluations per training run for mini-batch sizes of $|\mathcal{B}| \in \{10, 100, 1000\}$. In this investigation all parameters pertaining to GOLS-I and the construction of the training problem remain the same, with the exception of conducting both static and dynamic MBSS.

3.3.4 Naming convention

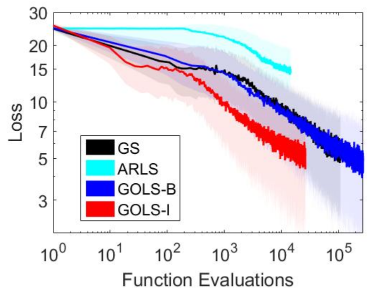
Since we are able to determine α_{n,I_n} for LS-SGD or a variety of different training algorithms that have learning rates, using a number of different line search methods, we augment the name of the training algorithm to accommodate the name of the method. For example, implementing GOLS-I to determine step sizes in LS-SGD is denoted "GOLS-I SGD". However, as all the numerical studies performed use LS-SGD, with the exception of one analysis in Section 3.4.5, we only explicitly use this naming convention in this chapter for the case where GOLS-I Adagrad is used in Section 3.4.5. In all other investigations LS-SGD is implied, and therefore the name of the line search used is distinguished.

3.4 Results

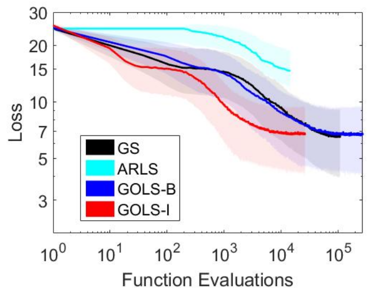
The results are split into different loss plots for the training, validation and test datasets respectively. This shows the training effectiveness of the various line searches, and displays the consistency of obtained solutions to the unseen data. It is important to note here, that since the optimizers operate only on the training data, their specific performances are gauged predominantly on the training data losses. The validation and test dataset loss plots indicate the generalization of the trained networks to unseen data, establishing the utility and degree of overfitting of a trained network as a secondary outcome.

3.4.1 Averaged results

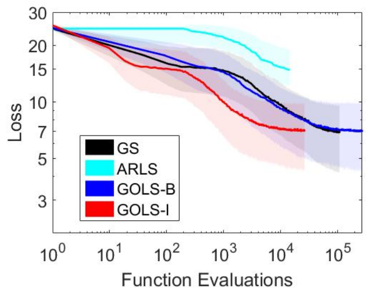
Figure 3.8 shows the loss plots as averaged over all the foundational datasets for their respective single hidden layer networks. The solid lines indicate the average losses for the different line search algorithms in terms of the number of function evaluations. The shaded regions indicate the variance clouds around the expected loss.



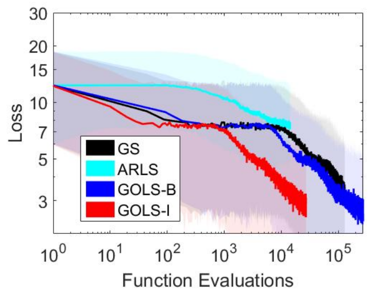
(a) Training data



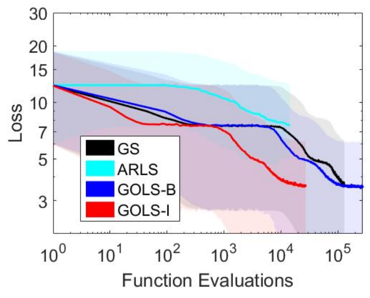
(b) Validation data



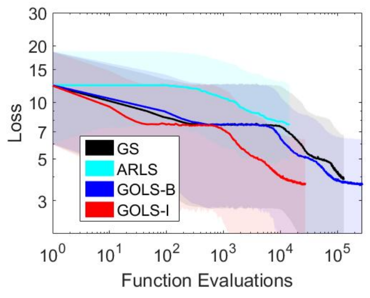
(c) Test data



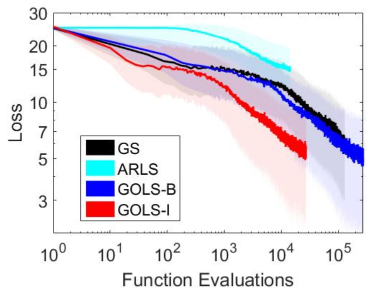
(d) Training data



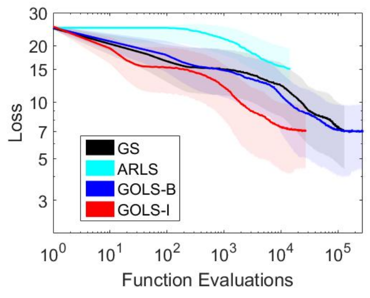
(e) Validation data



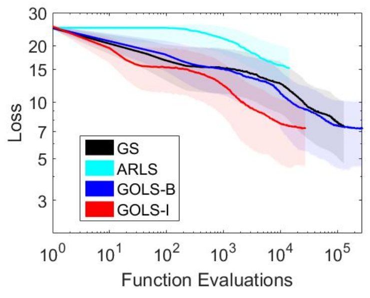
(f) Test data



(g) Training data



(h) Validation data



(i) Test data

Figure 3.8: Average training, validation and test dataset losses averaged over (a)-(c) single hidden layer networks, (d)-(f) double hidden layer networks and (g)-(i) both single and double hidden layer networks.

The large standard deviations in Figure 3.8, are indicative of a range of characteristics in the considered datasets. As expected, due to optimization being performed on only the training datasets, the training losses reduce more aggressively than the validation and test losses. It is clear, that on average the models begin to overfit. The characteristics between the validation and test plots are similar, which is a good indication that tuning of hyperparameters based on the validation set would be representative of performance on the "hold-out" test dataset. For example, should a stop criterion be implemented around 10^4 function evaluations for GOLS-I, this would also result in a good test dataset loss. This would stop training before overfitting drives both validation and training loss higher at wasted computational expenditure.

In terms of computational cost, the exact line search algorithms require more function evaluations per iteration than the inexact line searches, as expected. The gradient-only based line searches also require more function evaluations than the equivalent function value based methods. Therefore the question is whether the performance of the gradient-only methods outweighs their added computational cost.

For the exact line searches consider the case of GS in the single hidden layer average results in Figures 3.8(a)-(c). Though the training loss of GS does not progress as far down as that of GOLS-B, the validation and test losses are essentially the same. This may indicate an ability to find better local minima. However, this trend is not general, as can be seen in the case of the double hidden layer networks in Figures 3.8(d)-(f). In this case GOLS-B is superior in both loss values and computational cost to GS. This is evident for the averaged results over all problems in Figures 3.8(g)-(i).

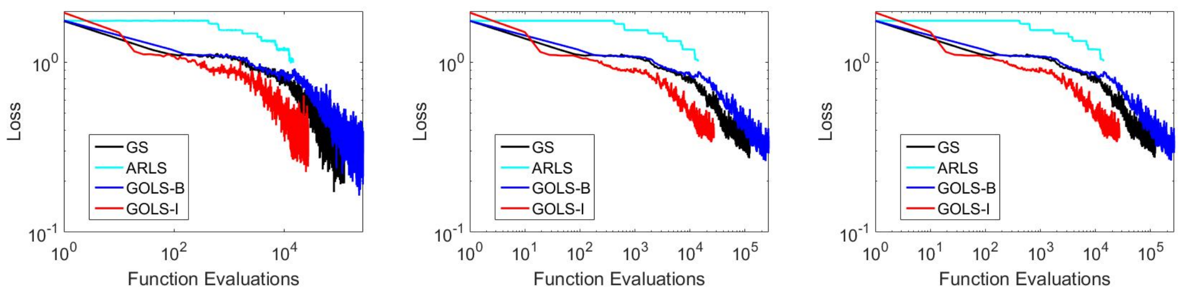
Between the inexact line searches it is clear that GOLS-I is superior. Function value based ARLS fails to converge to useful solutions within the given number of iterations. Notably, its performance is significantly worse than that of GS. This is due to the bracketing strategy of GS, that exponentially grows the bracketed domain. This usually ensures that significantly larger step sizes are taken than those of ARLS. Note, that though both methods use function value information to resolve minimizers, there is a significant difference in their exhibited performance, which again indicates that finding minimizers is not as robust as finding SNN-GPPs. This is evident when we consider the performance of the gradient-only based GOLS-I, which is comparable to GOLS-B at an order of magnitude lower number of function evaluations. This is consistent over all three average loss plots, demonstrating the ability of GOLS-I to be competitive over both different datasets as well as network architectures.

3.4.2 Examples of individual best performances for different methods

For the foundational datasets, there were specific datasets, where different methods performed considerably better than reflected in their averages. Such performances were measured by considering training, validation and test losses obtained at similar computational cost. If the losses for the different methods were comparable, the best performer is the method with the least computational cost. However, if a clear minimum in loss was achieved by a given method in the set number of iterations, that method was considered to be the best performer, regardless of its computational cost. Given here are some examples of such cases and the number of best performances per line search method.

Golden Section as best performer (2 out of 43 cases)

Figure 3.9 gives an example of GS being effective in resolving quality step sizes. The given problem is the Human Activity Recognition (HAR) dataset [Anguita et al., 2012] with a single hidden layer architecture. In the training data loss plot in Figure 3.9(a), the convergence rate of GS is comparable to that of GOLS-B, while being more computationally efficient. In contrast, GOLS-I seems to stagnate towards the end of training, especially when considering the validation and test dataset plots. Another case where GS was effective, is the Heartc1 dataset problem [Prechelt, 1994] with 2 hidden layers. However, for the remaining problems, performance of GS remained poor, being expensive and not offering competitive convergence performance compared

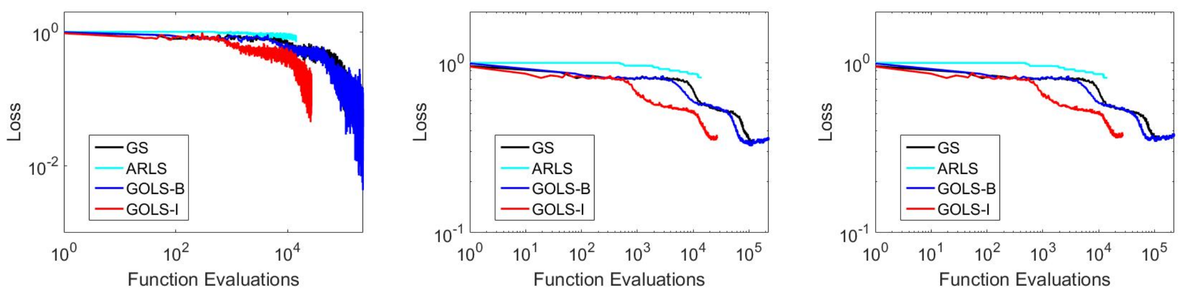


(a) Training data (b) Validation data (c) Test data

Figure 3.9: Average (a) training, (b) validation and (c) test dataset losses obtained for different line searches for the HAR dataset [Anguita et al., 2012] using the single hidden layer architecture, an example of a dataset where the function value based Golden-Section method was the best performer.

to the gradient-only methods.

Gradient-Only Line Search with Bisection as best performer (7 out of 43 cases)



(a) Training data (b) Validation data (c) Test data

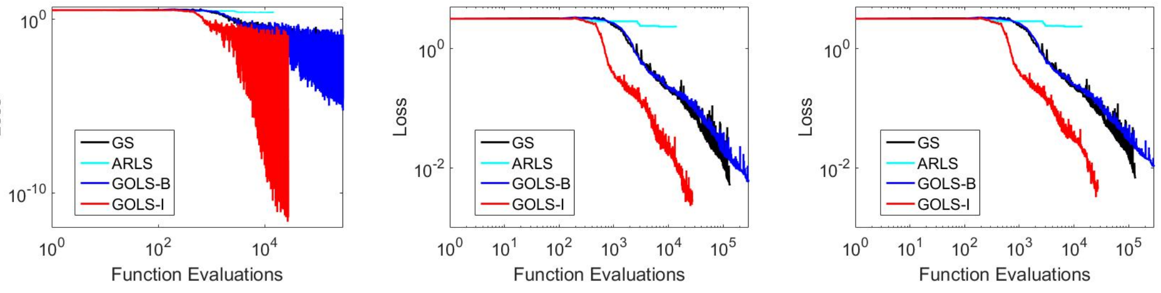
Figure 3.10: Average (a) training, (b) validation and (c) test dataset losses obtained for different line searches for the Gene1 dataset [Prechelt, 1994], an example of the Gradient-Only Line Search with Bisection as the best performer.

Figure 3.10 shows an example of GOLS-B being the most effective in training. The problem shown here is the Gene1 dataset [Prechelt, 1994], with 2 hidden layers. Its convergence rate is the fastest, obtaining the lowest training as well as validation and test losses for the given number of iterations. Though GOLS-I achieves similar training convergence rates at a reduced computational cost, it is GOLS-B, that achieves a better generalization loss for this problem. Examples such as these indicate that there are problems which require more accurate resolution of the SNN-GPP in order to find solutions with better generalization properties.

An analysis with equivalent computational cost between GOLS-I and GOLS-B would be required to determine an absolute performance comparison between the two. However, it is unlikely that GOLS-I would beat GOLS-B in generalization, since it already overfits in the given analysis. Nevertheless, given the number of fixed iterations, there were 7 datasets on which GOLS-B marginally better than the rest.

Gradient-Only Line Search that is inexact as best performer (34 out of 43 cases)

Overall, GOLS-I is the most efficient in reducing the training loss while maintaining a low computational cost. As confirmed in the averaged loss plots, it is able to reach comparable validation and test loss values compared to GOLS-B at an order of magnitude fewer function evaluations. However, in some cases its performance was considerably better than that of the rest. Figure 3.11 shows a case where the progress made is both more efficient as well as superior in convergence.



(a) Training data (b) Validation data (c) Test data

Figure 3.11: Average (a) training, (b) validation and (c) test dataset losses obtained for different line searches for the Mushroom1 dataset [Prechelt, 1994], an example of the gradient-only line search that is inexact method as the best performer.

For the Mushroom1 dataset [Prechelt, 1994] with the single hidden layer architecture, GOLS-I progresses quickly towards good solutions, causing the loss to drop rapidly. In training, the obtained loss is 5 orders of magnitude lower than the nearest competitor GOLS-B. GOLS-I also found a more general solution compared to GOLS-B and GS. This is one of the more extreme examples, where training is superior both in loss achieved as well as convergence rate. However, in the majority of cases the method was capable of returning comparable or better loss values than the remaining methods at a order of magnitude lower computational cost, which held for 34 out of 43 of the examined problems.

This shows that the method, though inexact, is capable of reliably estimating step sizes for various datasets and neural network architectures.

3.4.3 Comparing step size characteristics of the Glass1 and Cancer1 datasets

To investigate the adaptive nature of step sizes, the resolved step sizes at every iteration of the various line search methods are depicted in Figures 3.12(a)-(f). Depicted are the mean step sizes, training losses as well as validation losses averaged over 10 runs for the Glass1 and Cancer1 datasets with the double hidden layer architecture.

Consider the step sizes resolved for the Glass1 problem depicted in Figure 3.12(a). The function value based ARLS produces small step sizes and therefore does not progress throughout the loss function, as has been previously observed. The GS line search is more effective, as it produces step sizes with a reasonable magnitude. However, these seem to vary around a constant mean value. Conversely, GOLS-I and GOLS-B show a distinct variation in step sizes as a function of iterations, first increasing by an order of magnitude to around $\alpha_{n,I_n} = 10^1$ around 500 iterations, then slowly decreasing to around $\alpha_{n,I_n} = 10^0$. This indicates, that GOLS resolve step sizes adaptively for discontinuous loss functions that are difficult to pre-empt for a learning rate schedule.

In turn, the step sizes resolved for the Cancer1 problem in Figure 3.12(d), are significantly different to those of the Glass1 problem. GOLS-I's step sizes are distinct from those of GOLS-B, being 4 orders of magnitude larger and around the maximum allowed step size, $\alpha_{n,I_n} = 10^7$. This shows that GOLS-I can significantly overshoot SNN-GPPs resulting in larger step sizes, whilst GOLS-B resolves step sizes that are within the ball B_ϵ. The decrease in gradient magnitude towards the end of training results in GOLS-I increasing its step sizes towards the maximum allowed. These large step sizes that overshoot SNN-GPPs still relate to good performance on the training loss as depicted in Figure 3.12(e). In terms of validation loss, depicted in Figure 3.12(f), a decrease in generalization is observed due to overfitting on the dataset. However, GOLS-I still remains the best performer.

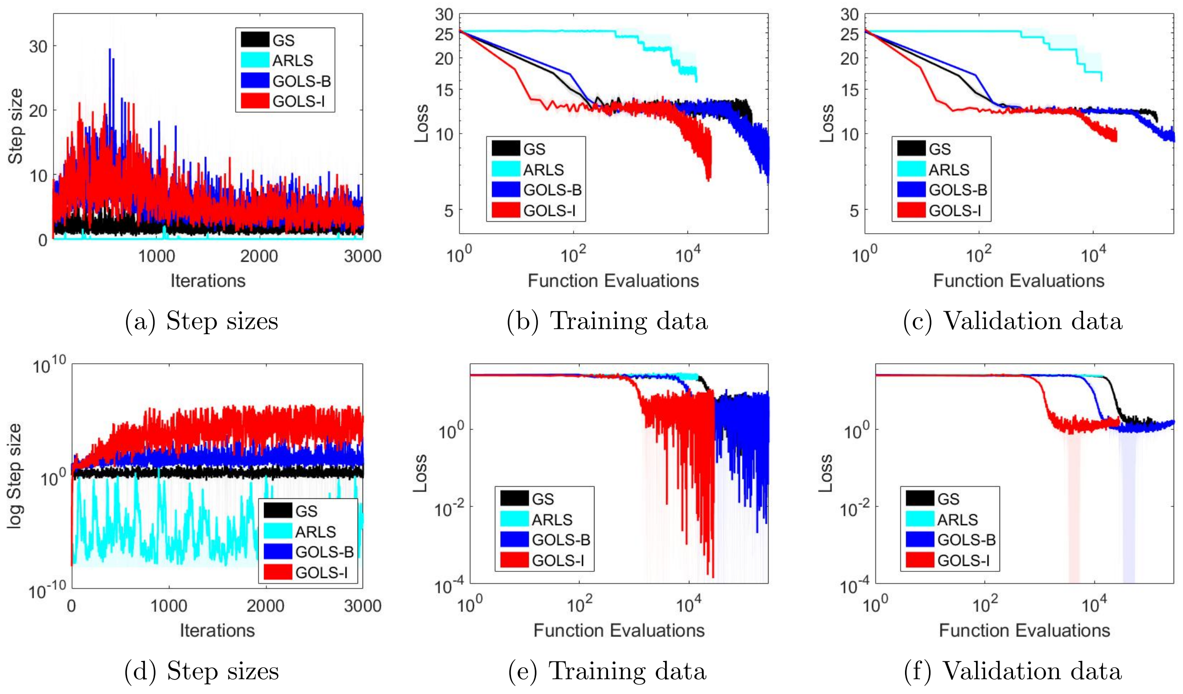


Figure 3.12: Step size investigations for (a)-(c) the Glass1 dataset [Prechelt, 1994] and (d)-(f) the Cancer1 dataset [Prechelt, 1994] used with the double hidden layer network architecture. Shown are average step sizes, training losses and validation losses obtained for different line search methods. The step sizes are given as a function of iterations, as step sizes are resolved on an iteration basis. The training and test losses are given in the function value domain to remain comparable with Figures 3.8-3.11. This illustrates that GOLS-I has comparable characteristics to GOLS-B in resolving step sizes, but at a fraction of the computational cost.

The poor performance of ARLS can be attributed to local minima that result from the discontinuous loss function, in particular, positive jump discontinuities [Wilke et al., 2013]. This significantly hampers resolving appropriate the step sizes away from its conservative initial guess. In the case of GS, an exponential bracketing strategy is used, which is less prone to this problem. It is the magnitude of the parameters in the bracketing strategy that aids the progress. The discontinuities cause the bracketing strategy to fall short of encompassing a true minimum. This results in GS not reliably adjusting the step size according to features in the loss function. Statistically, this makes GS perform similarly to a fixed learning rate, dependent on the magnitude of the parameters chosen for the initial bracketing strategy.

Conversely, both GOLS-B and GOLS-I are capable of adjusting the step size to the required magnitude within a single optimization iteration, as can be seen for both investigated cases in Figure 3.12(a) and (d). At the first iteration, both gradient based methods converge to similar step sizes magnitude. It is in the latter stages of training for the Cancer1 dataset that the two methods diverge in step size. The step size trends over iterations being distinct for the Glass1 and Cancer1 datasets underlines the importance of resolving the step size automatically on an iteration basis via effective line search strategies.

3.4.4 Analysis of iterative performance of line search methods

It is evident, that the computational cost per iteration varies between line searches. We quantify this variation by counting the average number of function evaluations per iteration for the line searches in this investigation, and summarize them in Table 3.3.

To give context, a single iteration of LS-SGD with a constant learning rate consists of 2 function evaluations, since the gradient needs to be computed at every iteration. Function value based line searches are essentially half as expensive as the gradient-only based methods,

	GS	GOLS-B	ARLS	GOLS-I
Ave. # of function evaluations	42.8	83.3	4.75	9.0
Ave. # of information calls	42.8	41.7	4.75	4.5

Table 3.3: Average number of function evaluations and information calls per iteration for the various line searches.

due to the added cost of computing the gradient. Inexact methods require around an order of magnitude less function evaluations as compared to their exact line search counterparts. Though comparing function evaluations per iteration gives an indication of computational cost of the methods, it only indirectly accounts for the number of times line searches required new information per iteration. We therefore also note the number of "information calls", where a function value and gradient evaluation each constitute a single information call. On this basis the respective exact and inexact line search methods perform similarly, having similar number of information calls. This shows that algorithmically these methods are comparable, relying on the same number of information per iteration. The only difference between these algorithms is merely the information they use. Even with the added computational cost of the gradient-only line searches, the information gain is substantial enough to offset its cost.

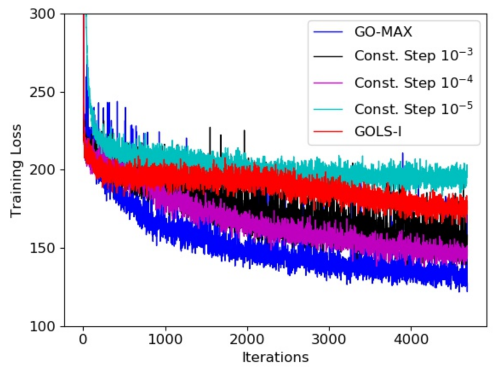
Due to the stochastic nature of the loss function, it seems more reasonable to consider inexact rather than exact strategies. An exact line search strategy wastes computational resources resolving the accuracy to a bound that is smaller than the variance in the solution due to stochastic sub-sampling. Gradient-only based GOLS-I is able to bypass discontinuities by observing consistent gradient trends in the loss function, while requiring fewer gradient evaluations than GOLS-B. It is therefore a plausible method to efficiently resolve the learning rate in the context of discontinuous loss functions, as a result of dynamic MBSS, to sufficient accuracy.

3.4.5 Variational Autoencoder Training Investigation

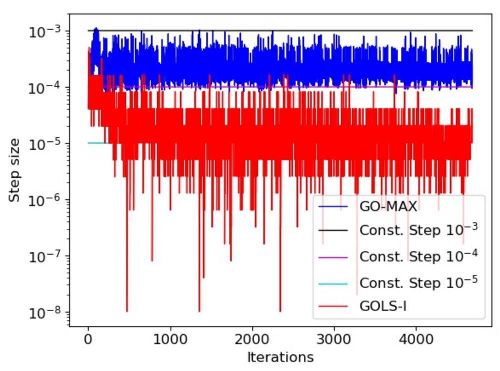
The resulting training loss and corresponding step sizes for the Variational Autoencoder (VAE) example are given in Figure 3.13. The training loss in Figure 3.13(a) shows the performance of the different training methods. We remind the reader, that LS-SGD is used in this investigation until otherwise stated. Therefore, labels in the figures denote the strategy used to determine step sizes. The fixed learning rates $\alpha_{n,I_n} = 10^{-3}$ and $\alpha_{n,I_n} = 10^{-5}$ are too large and too small respectively. This is indicated by the losses with $\alpha_{n,I_n} = 10^{-4}$ being lower than either of the other constant step sizes. This demonstrates the sensitivity of this problem to the learning rate. It is necessary to estimate the step size to within an order of magnitude in order to have satisfactory training performance.

Interestingly, the convergence of GOLS-I is superior to that of the fixed step size of $\alpha_{n,I_n} = 10^{-5}$, but worse than either of the larger constant step sizes. This led to the development of a modified formulation of GOLS-I, named GOLS-Max, which maximizes the step size and thereby aggressively biases overshooting the SNN-GPP (see Appendix A.3.3). For GOLS-Max, the step size is maximized to approximate a specified directional derivative overshoot. In our case we kept this the same as the upper bound of the immediate accept condition for GOLS-I, meaning that we choose the step size such that $\frac{dF(\alpha)}{d\alpha} \approx (c_2)|\frac{dF(0)}{d\alpha}|$. For our VAE example, this consistently results in step sizes being resolved that are about an order and a half larger than those resolved by GOLS-I, see Figure 3.13(b). Surprisingly, this caused a drastic improvement in training performance, see Figure 3.13(a). Note that for this investigation we consider all analyses in the iterations domain. This is to allow direct comparison of the quality of the information used in the different line search formulations, and not to scrutinize their comparative computational efficiency.

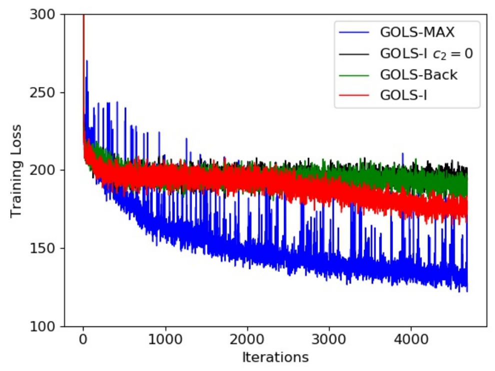
This behaviour poses the question as to whether one obtains the best convergence performance by finding optima, or whether these optima are beacons that need to be passed along the path to better solutions for this problem. To investigate this question we conducted a number of additional analyses. The first step is to put the performance of GOLS-I into context. We



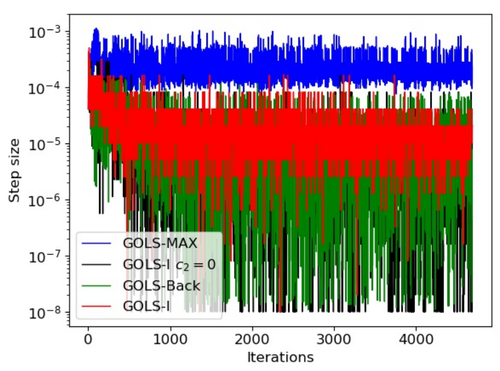
(a) Training Loss



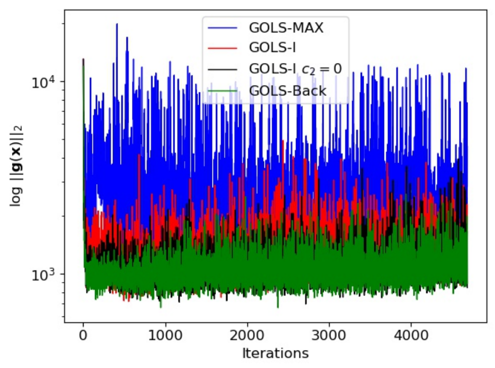
(b) Step Size



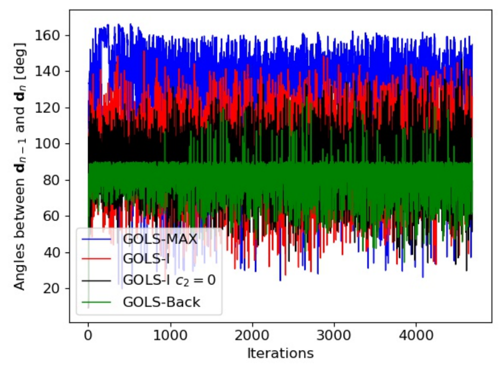
(c) Training Loss



(d) Step Size



(e) Gradient norms at α_{n,I_n}



(f) Angle between successive descent directions

Figure 3.13: (a) Training loss and (b) step sizes for training a Variational Autoencoder on MNIST [Lecun et al., 1998] using four variants of GOLS-I. GOLS-I estimates step sizes that are small compared to constant step sizes and suffers worse convergence performance. Consequently we implement GOLS-Max (which is more competitive), GOLS-I with $c_2 = 0$ (no immediate accept condition) and GOLS-Back (always accepts first negative direction derivative) in (c) and (d), with (e) and (f) indicating the norm of descent directions and angles between successive descent directions respectively.

conduct another analysis with GOLS-I, but set the overshoot parameter to $c_2 = 0$. This constitutes a conservative line search approach, as the immediate accept condition becomes inactive. To ensure that this conservative line search does not simply find local optima that are close the initial guess, a gradient-only backtracking line search, GOLS-Back is also introduced (see Appendix A.3.4). This is a more aggressive algorithm to find a sign change, as its initial guess is the maximum step, which is reduced until a negative sign change is encountered. Consider see Figure 3.13(c) and (d), which shows a comparison between the training performance and

	GOLS-Max	**GOLS-I**	**GOLS-I $c_2 = 0$**	**GOLS-Back**
mean angle [deg]	135	101	89	81

Table 3.4: Mean angles between consecutive search directions $\boldsymbol{d}_{n-1}$ and $\boldsymbol{d}_n$ for various gradient-only line search methods with LS-SGD.

step sizes for the different GOLS formulations. The fact that both the conservative GOLS-I and GOLS-Back do not differ significantly in performance or in resolved step size, indicates three important aspects: Firstly, the line searches are indeed approximating the location of SNN-GPP for this problem (two distinct formulations show similar results); secondly, the local minima along the descent directions are not significantly far apart, meaning the loss function seems unimodal along the descent direction; and lastly, finding these optima along the descent direction does not result in good convergence performance. Indeed, better performance can be obtained by overshooting these optima. So what is the mechanism driving the sub-optima performance of gradient-only line searches? The quality of descent directions.

To substantiate this claim, consider the following investigation: For the same four versions of GOLS, we capture the norm of the gradient at the end of the line search (or descent direction of the subsequent iteration), as well as the angle between successive descent directions in Figure 3.13(e) and (f). The norms of the gradients at line search termination are considerably larger for GOLS-Max than for the rest of the methods. This is consistent with the concept of overshooting the SNN-GPP, since the resolved solution in on an ascending incline with larger gradients. The remaining methods are in a less steep domain, indicated by smaller gradients, which is consistent with being closer to a SNN-GPP. As expected, conservative GOLS-I and GOLS-Back have similar behaviour, resolving SNN-GPP most accurately, resulting in the smallest gradient norms. The slight overshoot of GOLS-I is also evident, as its resolved norms tend towards higher values than those of conservative GOLS-I and GOLS-Back. However, Figure 3.13(f) is the most telling as far as the observed convergence behaviour is concerned. All methods apart from GOLS-Max have successive descent direction angles centred around 90 degrees. We have explicitly listed the average angles throughout the given training duration in Table 3.4. Successive directions of 90 degrees are a typical trait of steepest gradient descent with a reasonably accurately resolved line search [Arora, 2011]. Since LS-SGD is being used, this similarity is impressive, since it indicates that firstly, the stochastically sampled search directions are on average representative of the full-batch directions for this batch size; and secondly, the step sizes are reasonably accurate. The visual results of GOLS-Back in Figure 3.13(f) are telling, as they are closely clustered around 90 degrees. Since GOLS-Back does not allow any overshoot, the odds of having a successive search direction that points to some degree back towards the opposite direction (angle $> 90\,\text{deg}$) are less likely, which corresponds to what we observe. In contrast, GOLS-Max has a far higher average angle than the remaining methods, indicative of back and forth oscillation within the loss function.

An explanation for why this still results in competitive convergence performance is given in the following analogy: Suppose the loss function is a narrow ravine, much like a slanted pipe cut in half along its length, as illustrated in Figure 3.14. The steepest descent direction will point radially predominantly in towards the centre of the pipe, with a smaller component pointing along its length in the general descent direction. If an inexact methods attempts to resolve the optimum along the search direction, a degree of error is incurred, resulting in an over- or undershoot. If the curvature is still steep at that point, an overshoot will cause the subsequent descent direction to point predominantly back towards the centre of the pipe. This property is exacerbated, depending on the degree of overshoot, resulting in a large angle between subsequent descent directions. If resolving the optimum is inaccurate, but not severely over- or undershot, the subsequent descent direction is almost perpendicular to the previous, as has been observed for the steepest descent algorithm [Arora, 2011]. However, it is not the larger components of the descent direction (pointing toward the centre of the pipe) that dominate progress here, but the smaller components, that point along the pipe. A large step along the dominant components also imply a larger step along the smaller components. As shown in Figure 3.14, the overshoot

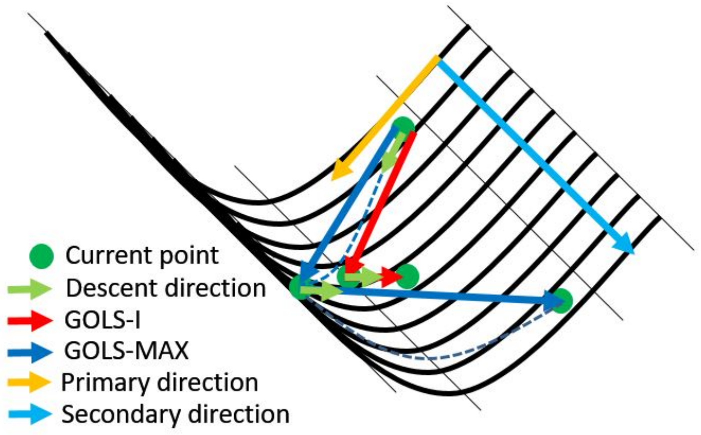


Figure 3.14: Gradient descent optimization along narrow, "pipe"-like valleys can be decomposed into primary and secondary directions of improvement. Larger step sizes result in significant oscillations in primary directions of improvement, but ultimately lead to faster progression along secondary directions.

allows a large degree of travel up the opposite side of the pipe, but also a significant step along the slant of the pipe. The subsequent step "bounces" back along the steep incline of the pipe, but also again progresses along the slant of the pipe. During the oscillation along the pipe, the sum of the larger components are eliminated due to pointing in opposite directions, but the smaller components along the pipe accumulate for progression along the pipe. It is the large accumulation of these smaller components that dominate the convergence performance. This is the same argument made for second order methods in machine learning [Martens, 2010], the implementations of which require robust line searches in dynamically sub-sampled loss functions.

We substantiate the directions argument by applying GOLS-I to resolve the learning rate of the Adagrad algorithm [Duchi et al., 2011], which we subsequently refer to as GOLS-I Adagrad. The directions given by this algorithm are updated based on historical gradient information and therefore loosely translates to a variable metric method. An overview of the method can be given as follows:

$$h_{n,q} = -\tilde{g}(\boldsymbol{x}_n)_q, \tag{3.19}$$

$$x_{n+1,q} = x_{n,q} + \frac{\beta}{\sqrt{H_{n,qq} + \gamma}} \cdot h_{n,q}, \tag{3.20}$$

for the q^{th} component of the p-dimensional optimization problem, β is a constant learning rate and γ is a small constant, usually of the order 10^{-8}, added for numerical stability. H_n is a diagonal matrix, where the entries denote the sum of squares of historical gradient components, $H_{n,pp} = \sum_{t=1}^{n} h_{t,p}^2$. Therefore, historical information contained in the components of $H_{n,pp}$, scale the current gradient, augmenting the descent direction. The inverted relationship to H_n means that low gradient magnitude components (the slant of the "pipe") are amplified, while large magnitude components (the walls of the "pipe") are reduced, in essence making the "pipe" appear more spherical.

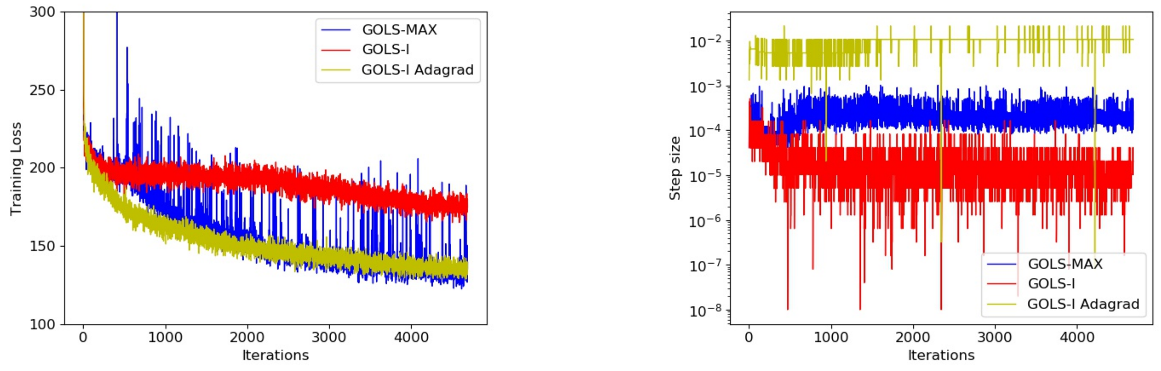
Based on the Adagrad formulation, we define the descent direction, $\boldsymbol{d}_n$ for the update at iteration, n, with individual components

$$d_{n,q} = -\frac{h_{n,q}}{\sqrt{H_{n,qq} + \gamma}}, \tag{3.21}$$

and define the parameter update step as

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_{n,I_n}\boldsymbol{d}_n, \tag{3.22}$$

where we are subsequently able to resolve the learning rate parameter as a step size, α_{n,I_n}, using GOLS-I. Adagrad is just one example of various methods used in machine learning, that scale the search directions to be more uniform, allowing the optimizer to more closely follow the contours of the loss function [Ruder, 2016]. Thus, by estimating Adagrad step sizes using GOLS-I, we obtain GOLS-I Adagrad, which is able to make more efficient progress through the loss function landscape.



(a) Training Loss (b) Step Size

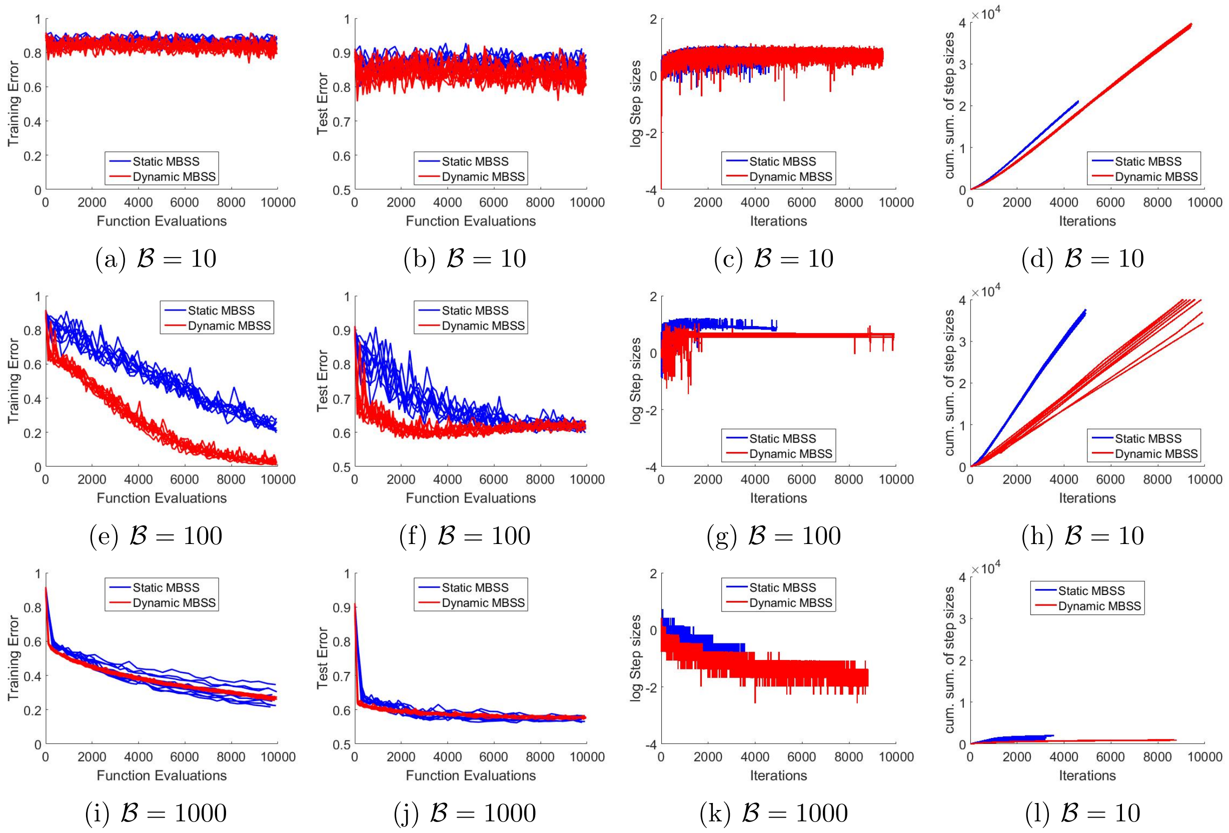
Figure 3.15: (a) Training loss and (b) step sizes for training a Variational Autoencoder on MNIST [Lecun et al., 1998], comparing LS-SGD with different inexact GOLS, to GOLS-I Adagrad, which uses scaled search directions. Convergence performance dramatically improves over LS-SGD and larger step sizes can be estimated by GOLS-I Adagrad.

The comparative performance is shown in Figure 3.15(a) and (b). It is clear that with LS-SGD, GOLS-I is not competitive and overshoot is required to improve performance. However, the improved directions of Adagrad make a significant difference. From the step sizes we also recognize that GOLS-I is able to resolve significantly larger steps, which corresponds to travelling along better descent directions ”down the slanted pipe”, to be consistent with our analogy. Though this analysis opens the discussion to using gradient-only line searches to resolve learning rates for improved descent directions (such as conjugate gradient and Quasi-Newton methods), extensive investigations are outside the scope of our consideration and are reserved for future work.

We have extensively shown that the use of SNN-GPPs in the context of line searches is a powerful tool to automatically resolve step sizes for discontinuous, dynamic MBSS loss functions, prevalent in neural network training. We introduced a number of gradient-only line search methods: GOLS-B, GOLS-I, GOLS-Back and GOLS-Max; as alternatives to locate SNN-GPPs. Of these we suggest GOLS-I as the default method, as it balances low computational cost with moderate aggression in terms of step size, and ease of implementation. Gradient-only based GOLS-I is capable of a step size range over 15 orders of magnitude and was immediately able to resolve competitive learning rates automatically. Our investigations confirm the utility of GOLS-I in eliminating the learning rate hyperparameter from neural network training while using dynamic stochastic sub-sampling.

3.4.6 A demonstration of the merits of dynamic mini-batch sub-sampling

Recall the standard SGD update step, $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \alpha_{n,I_n}\boldsymbol{g}(\boldsymbol{x}_n)$ [Robbins and Monro, 1951]. Adaptive sampling methods [Bollapragada et al., 2017, Friedlander and Schmidt, 2011] focus predominantly on desired qualities of the $\boldsymbol{g}(\boldsymbol{x}_n)$ term. This leaves the step size, α_{n,I_n}, as an unresolved parameter. In addition, numerous researchers have speculated over the benefits of resampling the mini-batch at every function evaluation [Bottou, 2010, Mahsereci and Hennig, 2017] (i.e. dynamic MBSS), due to a larger throughput of data. A concrete study based on this observation within in the context of resolving learning rates has remained elusive. This has been



(a) $\mathcal{B} = 10$ (b) $\mathcal{B} = 10$ (c) $\mathcal{B} = 10$ (d) $\mathcal{B} = 10$

(e) $\mathcal{B} = 100$ (f) $\mathcal{B} = 100$ (g) $\mathcal{B} = 100$ (h) $\mathcal{B} = 10$

(i) $\mathcal{B} = 1000$ (j) $\mathcal{B} = 1000$ (k) $\mathcal{B} = 1000$ (l) $\mathcal{B} = 10$

Figure 3.16: Static vs. dynamic MBSS: (a,e,i) Training and (b,f,j) test dataset classification errors in terms of function evaluations for mini-batch sample sizes, $|\mathcal{B}| \in \{10, 100, 1000\}$, with (c,g,k) estimated step sizes and (d,h,l) corresponding cumulative step sizes. This problem is adopted from [Mahsereci and Hennig, 2017] and implements a 3-hidden-layer network to classify the CIFAR10 test dataset, while training on Batch1 of the training dataset. This problem is limited in the number of function evaluations, resulting in different numbers of performed iterations, based on the efficiencies of the line searches.

mainly due to the lack of an effective mathematical and optimality framework to resolve information for dynamic MBSS discontinuous loss functions. Having proposed an optimality framework that has been demonstrated to be robust in this study, allows us to investigate whether there is a concrete benefit in conducting dynamic MBSS over static MBSS when resolving learning rates. As employed by Mahsereci and Hennig [2017], dynamic MBSS allows for the utilization of the last computed gradient at the previous iteration to serve as the search direction for the next iteration. Conversely, static MBSS requires the search direction to be recomputed for each iteration so that it is consistent with the current mini-batch.

The results, shown in Figures 3.16(a) and (b), indicate that both line searches are ineffective for the smallest batch size, $|\mathcal{B}| = 10$. For $|\mathcal{B}| = 100$ presented in Figures 3.16(e) and (f), the methods distinguish themselves in terms of both quality and computational efficiency, whereas for $|\mathcal{B}| = 1000$ the methods' performances are similar, as shown in Figures 3.16(i) and (j). Considering $|\mathcal{B}| = 100$, for both the training and test errors, dynamic MBSS GOLS-I outperforms static MBSS GOLS-I. This implies that dynamic MBSS GOLS-I is able to find better solutions on the training dataset, which also generalize better, as is evident from the test error. In both cases, the notable difference between the training and test errors point to overfitting on Batch1 of CIFAR10 during training.

By considering the differences in step sizes and cumulative step sizes between the static and dynamic MBSS GOLS-I approaches, presented in Figures 3.16(c), (d), (g), (h), (k) and (l), we can infer how the MBSS approach affects the GOLS-I step sizes per iteration. Since static MBSS GOLS-I has to evaluate a new search direction at every iteration, it completes under half the number of iterations for a fixed number of function evaluations compared to dynamic MBSS GOLS-I. It is evident that static MBSS GOLS-I takes larger steps than dynamic MBSS GOLS-I for the three batch sizes under consideration. Note that differences in step sizes are a direct consequence of the typical locations of sign changes along a descent direction, since GOLS-I is guaranteed to step over a sign change for both the static and dynamic MBSS loss functions. Hence, static MBSS sign changes are more likely to occur at larger step sizes along the descent direction compared to dynamic MBSS sign changes. This implies that the sign change of the mini-batch used to compute the fixed descent direction manifests at a larger step size than the sign changes resulting from other mini-batches being sampled along the same direction. As the batch size increases, the sign changes resulting from static and dynamic MBSS loss functions cluster closer together, until they coincide for the full-batch loss function. This seems to be the dominant factor contributing towards the similar cumulative step sizes between static and dynamic MBSS GOLS-I for $|\mathcal{B}| = 1000$. Placing the above observations into context, using a higher throughput of information [Bottou, 2010] to resolve step sizes along a fixed step direction, seems to result in more conservative line search updates. Although this preliminary inquiry clearly demonstrates the benefits of dynamic MBSS, it is by no means exhaustive and warrants further investigation into the interplay of static versus dynamic MBSS when resolving learning rates.

3.5 Conclusion

Currently, learning rates or learning rate schedules in ANN training are either selected *a priori* and manually adjusted until settings are found that are deemed acceptable by the user, or expensively solved at the global hyperparameter level. Attempts to resolve step sizes using line searches have proven to be challenging. This is mainly due to dynamic mini-batch sub-sampling (MBSS) that is readily used when optimizing the weights for neural networks, to speed up function evaluations for large datasets, save memory to allow computation on memory limited devices such as graphical processing units (GPUs) and maximize the throughput of training samples. However, dynamic MBSS comes at the cost of introducing discontinuities in the loss function, which introduces local minima that may significantly hamper a function value minimization line search strategy. This is evident by the poor performance of both the exact and inexact function value based line searches in our investigations, when dynamic MBSS is used. However, dynamic MBSS does have the benefit of adding variance to the function values and gradients, which allows optimization strategies to move beyond ”weak” local minima or SNN-GPPs in the loss function.

This chapter demonstrated that line searches can be reliably performed to determine step sizes in discontinuous loss functions as seen in Neural Network training, alleviating the need for *a priori* determined step sizes or step size rules. However, this required changing the optimizer aim from minimizing along a search direction to finding stochastic non-negative gradient projection points (SNN-GPPs) along a search direction. This is based on the observation, that local minima due to sampling discontinuities do not manifest so readily as SNN-GPPs. In addition, SNN-GPPs can be readily resolved using gradient-only line search (GOLS) strategies. This chapter demonstrated that in the context of discontinuous loss functions due to dynamic MBSS, the learning rate in steepest gradient descent can be reliably resolved using exact or inexact gradient-only line search strategies over 22 different datasets used in shallow, deep, and Variational Autoencoder neural network architectures with different loss functions.

Training of the Variational Autoencoder neural network did not seem to benefit from locating SNN-GPPs. This is due to the poor descent direction utilized in SGD algorithms, in which significant performance gains can be achieved by maximizing the step size along a descent direction as opposed finding the optimal step lengths along a descent direction. As demonstrated, by subsequently resolving SNN-GPPs along improved search directions, the performance increased significantly. This observation naturally extends towards the potential benefits of GOLS when considering conjugate gradient and Quasi-Newton approaches within the context of dynamic

MBSS neural network training.

Being able to robustly resolve step sizes for dynamic MBSS discontinuous loss functions enabled us for the first time, to conduct a preliminary investigation into the potential benefits of having a higher throughput of information from the training dataset when resolving learning rates. Our findings clearly demonstrate the benefits of dynamic MBSS from a computational and quality of solution viewpoint. In addition, we showed that dynamic MBSS step sizes tend to be more conservative when compared to step sizes resolved using a static MBSS approach.

Chapter 4

Gradient-only line searches: An Alternative to Probabilistic Line Searches

Step sizes in neural network training are largely determined using predetermined rules such as fixed learning rates and learning rate schedules. These require user input or expensive global optimization strategies to determine their functional form and associated hyperparameters. Line searches are capable of adaptively resolving learning rate schedules. However, due to discontinuities induced by mini-batch sub-sampling, they have largely fallen out of favour. Notwithstanding, probabilistic line searches, which use statistical surrogates over a limited spatial domain, have recently demonstrated viability in resolving learning rates for stochastic loss functions.

This chapter considers an alternative paradigm, Gradient-Only Line Searches that are Inexact (GOLS-I), as an alternative strategy to automatically determine learning rates in stochastic loss functions over a range of 15 orders of magnitude without the use of surrogates. We show that GOLS-I is a competitive strategy to reliably determine step sizes, adding high value in terms of performance, while being easy to implement.

4.1 Introduction

Determining the learning rate, or learning rate schedule parameters, is still an active field of research in deep learning [Smith, 2015, Orabona and Tommasi, 2017, Wu et al., 2018], as these parameters have been shown to be the most sensitive hyperparameters in neural network training [Bergstra and Bengio, 2012]. In practice, their magnitudes are often selected *a priori* by the user. If these parameters result in update steps that are too small, training is stable, but computationally expensive. Conversely, with updates that are too large, training becomes unstable. In mathematical programming learning rates (step sizes) are commonly automatically resolved by line searches [Arora, 2011], see Figure 4.1. However, these traditionally require smooth and continuous loss functions on which to operate, see the full-batch loss in Figure 4.2.

In modern deep learning tasks the dataset sizes exceed the memory capabilities of individual computational nodes. In particular with the rise of memory-limited parallel computing platforms such as graphical processing units (GPUs) in deep learning, it is infeasible to evaluate full-batch loss functions. Instead, a mini-batch of available training data is sub-sampled to evaluate an approximate loss function. In fields such as adaptive sub-sampling methods, the primary concern is to resolve approximate loss functions with desired characteristics. The aim might be to select a sub-sample that results in the best mini-batch approximation of the full-batch loss function, or to ensure that selected approximate losses on average result in descent directions, [Friedlander and Schmidt, 2011, Bollapragada et al., 2017]. This means that for a standard update step, which consists of a search direction and corresponding step size, the focus of adaptive sampling methods is primarily on solving for the quality of the search direction. To make the most of the carefully selected mini-batch, it is kept constant while conducting a line search along the given

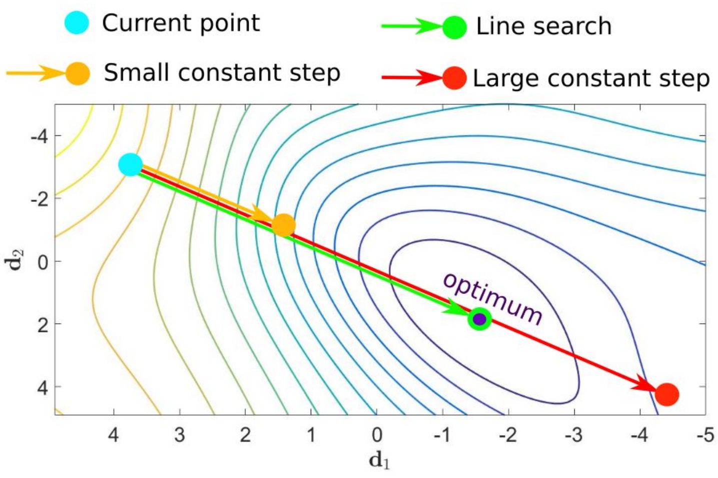


Figure 4.1: Contour plot of a neural network loss along two random perpendicular directions. Finding an optimum along a search direction: Too small constant steps are expensive, while too large constant steps can be unstable during training. Line searches balance performance and stability.

search direction [Byrd et al., 2011, 2012, Martens, 2010]. We call this approach static mini-batch sub-sampling (MBSS), see Figure 4.2, which constructs different loss function surfaces for every static mini-batch. Critically, this means that the loss function presented to a line search is continuous and smooth, which allows for the use of minimization line searches. However, the act of fixing the mini-batch introduces a sampling error, which biases the step size to the given mini-batch. The consequence is, that a different mini-batch can lead a line search to finding an alternative minimum, as demonstrated by the cyan surface in Figure 4.2.

An alternative is to continuously resample new data for every approximate loss evaluation. This approach was first introduced within the context of line searches by Mahsereci and Hennig [2017], though at the time not explicitly differentiated from static MBSS as used in adaptive sampling methods. We call the method of repeatedly sampling a new mini-batch for every function evaluation along a search direction *dynamic MBSS*, also known as approximate optimization [Bottou, 2010]. However, this spoils the utility of minimization line searches in neural network training, as randomly alternating between the sampling error associated with each mini-batch introduces discontinuities in resulting loss functions and gradients, see Figure 4.2. The consequence is that alternating between approximate loss expressions, critical points may not exist, and line searches falsely identify discontinuities as local minima [Wilson and Martinez, 2003, Schraudolph and Graepel, 2003, Schraudolph et al., 2007]. This led to line searches being replaced by *a priori* rule based step size schedules typical of sub-gradient methods, including stochastic gradient descent (SGD) [Schraudolph, 1999, Boyd and Park, 2014, Smith, 2015].

Recently, the use of Gaussian process surrogates using both function value and gradient information along search directions has reintroduced line searches to neural network training [Mahsereci and Hennig, 2017]. However, we postulate that a simpler and more accessible approach may be sufficient to construct line searches, using only gradient information. The premise for this proposition is that 1) a stochastic adaptation to the gradient-only optimality formulation of the Non-Negative Associated Gradient Projection Point (NN-GPP) [Wilke et al., 2013, Snyman and Wilke, 2018], i.e. the stochastic NN-GPP (SNN-GPP) [Kafka and Wilke, 2019b], is superior to local minima in identifying true optima in stochastic loss functions and 2) the discontinuities in function values are more severe than those in directional derivatives [Mahsereci and Hennig, 2017, Kafka and Wilke, 2019a], aiding SNN-GPP identification.

We demonstrate how the combination of these characteristics allows for the construction of gradient-only line searches that automatically determine step sizes. Based on empirical evidence, we argue that developing line searches that locate SNN-GPPs offers a competitive, light-weight and more robust alternative to conducting probabilistic line searches. We also present three

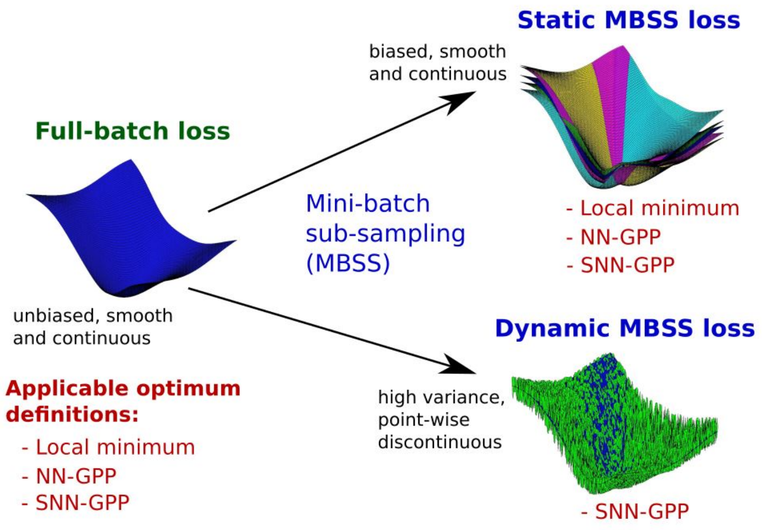

Static MBSS loss
biased, smooth and continuous
Full-batch loss
Mini-batch sub-sampling (MBSS)
- Local minimum
- NN-GPP
- SNN-GPP
unbiased, smooth and continuous
Dynamic MBSS loss
Applicable optimum definitions:
- Local minimum
- NN-GPP
- SNN-GPP
high variance, point-wise discontinuous
- SNN-GPP

Figure 4.2: Full and reduced fidelity loss representations with various definitions used to identify optima. Mini-batch sub-sampling (MBSS) can be conducted using static and dynamic approaches, resulting in smooth and point-wise discontinuous loss approximations respectively. The local minimum, non-negative associated gradient projection point (NN-GPP) and stochastic NN-GPP (SNN-GPP) [Kafka and Wilke, 2019b] definitions apply to full-batch and static MBSS losses. However, only the SNN-GPP is effective in localizing optima in dynamic MBSS loss functions.

examples of using gradient-only line searches as a research tool.

4.1.1 Our Contribution

In this chapter, we compare the *Gradient-Only Line Search that is Inexact* (GOLS-I) to the probabilistic line search (PrLS) proposed by Mahsereci and Hennig [2017] for determining learning rates in dynamically mini-batch sub-sampled loss functions. GOLS-I approximates the location of Stochastic Non-Negative Associated Gradient Projection Points (SNN-GPP) [Kafka and Wilke, 2019b], an adaptation of the gradient-only optimality criterion [Wilke et al., 2013, Snyman and Wilke, 2018]. When considering univariate functions, such as loss functions along a search direction, a SNN-GPP manifests as a sign change from negative to positive in the directional derivative along the descent direction. We stress that we do not rely on the concept of a critical point [Arora, 2011] as we do not require the derivative at a SNN-GPP to be zero. Specifying a sign change from negative to positive and not from positive to negative along a descent direction, incorporates second order information in that it reflects a local minimum as determined by gradient information.

Some common learning rate schedules use step sizes ranging over 5 orders of magnitude [Senior et al., 2013], while the magnitudes of cyclical learning rate schedules typically range over 3 to 4 orders of magnitude [Smith, 2015, Loshchilov and Hutter, 2016]. Manually selected schedules can require a number of hyperparameters to be determined. Our proposed method, GOLS-I, can determine step sizes over a range of 15 orders of magnitude without the need for any parameter tuning. The high range of available step sizes within the line search allow GOLS-I to effectively traverse flat planes or steep declines in discontinuous stochastic loss functions, while requiring no user intervention.

We also use this platform to 1) explicitly compare static and dynamic MBSS in the context of line searches, 2) uncouple the quality of search directions from the accuracy of resolving optima along the given direction and 3) demonstrate the sensitivity of search directions in SGD to mini-batch size in comparison to full-batch descent directions.

4.2 Loss Functions and means of locating optima

Commonly, the loss functions used in neural network training have the form

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

where $\{\boldsymbol{t}_1, \ldots, \boldsymbol{t}_M\}$ is a training dataset of size M, $\boldsymbol{x} \in \mathcal{R}^p$ is an p-dimensional vector of model parameters, and $\ell(\boldsymbol{x};\ \boldsymbol{t})$ defines the loss quantifying the fitness of parameters $\boldsymbol{x}$ with regards to training sample $\boldsymbol{t}$. Backpropagation [Werbos, 1994] computes the exact gradient w.r.t. $\boldsymbol{x}$, resulting in:

$$\nabla\mathcal{L}(\boldsymbol{x}) = \frac{1}{M}\sum_{b=1}^{M} \nabla\ell(\boldsymbol{x};\ \boldsymbol{t}_b). \tag{4.2}$$

In this case, all the training data is used for both function and gradient evaluations, resulting in the true or full-batch loss, $\mathcal{L}(\boldsymbol{x})$, and gradients, $\nabla\mathcal{L}(\boldsymbol{x})$, which are continuous and smooth. However, in modern deep learning problems the cost of computing the full-batch loss, $\mathcal{L}(\boldsymbol{x})$, is high. Therefore, mini-batch sub-sampling (MBSS) can be introduced to generate loss function approximations with using a subset of the training data, $\mathcal{B} \subset \{1, \ldots, M\}$ of size $|\mathcal{B}| \ll M$, resulting in:

$$L(\boldsymbol{x}) = \frac{1}{|\mathcal{B}|}\sum_{b\in\mathcal{B}} \ell(\boldsymbol{x};\ \boldsymbol{t}_b), \tag{4.3}$$

and corresponding approximate gradient

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

The approximate loss function has expectation $E[L(\boldsymbol{x})] = \mathcal{L}(\boldsymbol{x})$ and corresponding expected gradient $E[\boldsymbol{g}(\boldsymbol{x})] = \nabla\mathcal{L}(\boldsymbol{x})$ [Tong and Liu, 2005], but individual instances may vary significantly from the mean. Evaluating $L(\boldsymbol{x})$ instead of $\mathcal{L}(\boldsymbol{x})$ decreases the computational cost and increases the chance of an optimization algorithm overcoming local minima.

To formally extend our discussion to line searches, consider the following notation: We define a univariate function at given iteration n of stochastic gradient descent that uses a line search (LS-SGD) [Kafka and Wilke, 2019b] as $F_n(\alpha)$ along a descent direction, $\boldsymbol{d}_n \in \mathcal{R}^p$ from $\boldsymbol{x}_n \in \mathcal{R}^p$:

$$F_n(\alpha) = f(\boldsymbol{x}_n(\alpha)) = L(\boldsymbol{x}_n + \alpha\boldsymbol{d}_n), \tag{4.5}$$

with associated directional derivative

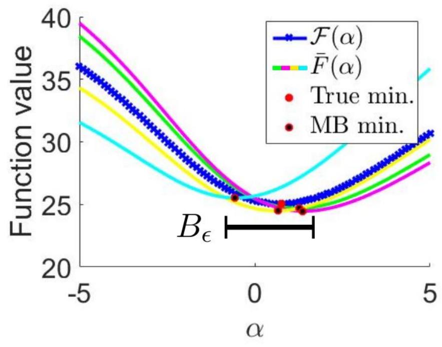
$$F_n'(\alpha) = \frac{dF_n(\alpha)}{d\alpha} = \boldsymbol{d}_n \cdot \boldsymbol{g}(\boldsymbol{x}_n + \alpha\boldsymbol{d}_n). \tag{4.6}$$

If full-batch sampling is implemented, the univariate loss representations along a search direction $\boldsymbol{d}_n$ is denoted $\mathcal{F}_n(\alpha)$ with respective derivative $\mathcal{F}_n(\alpha)$. Suppose static mini-batch sub-sampling (MBSS) is implemented, where:

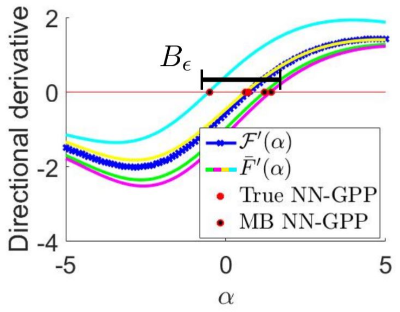
Definition 4.2.1. *Static mini-batch sub-sampling is conducted when the mini-batch, $\mathcal{B}$, used to evaluate approximations $L(\boldsymbol{x})$ and $\boldsymbol{g}(\boldsymbol{x})$ remains constant for a minimum duration of conducting a line search along a fixed search direction in iteration n of a training algorithm. Therefore, mini-batches selected using static MBSS are denoted as $\mathcal{B}_n$. The overhead bar notation is used to identify approximations evaluated using static mini-batch sub-sampling as $\bar{L}(\boldsymbol{x})$ and $\bar{\boldsymbol{g}}(\boldsymbol{x})$ respectively. [Kafka and Wilke, 2019b]*

The static MBSS loss is substituted into Equations (4.5) and (4.6) give static MBSS univariate functions $\bar{F}_n(\alpha)$ and $\bar{F}'_n(\alpha)$ respectively.

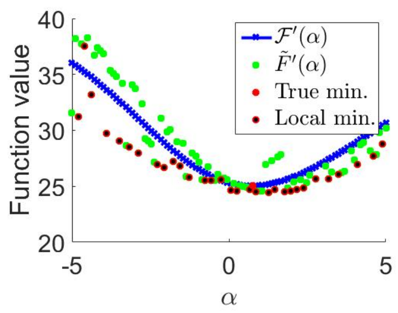
In Figure 4.3 we use a simple single hidden layer neural network with Sigmoid activation functions applied to the famous Iris [Fisher, 1936] dataset as an example of a loss function used in neural network training. Note, that subscript n is dropped, as the plots represent a single search direction. This allows us to explore loss function characteristics and applicable optimality criteria typically encountered during neural network training. Shown in blue in Figures 4.3(a) and (b) are $\mathcal{F}(\alpha)$ and directional derivative $\mathcal{F}'(\alpha)$ for the full-batch loss evaluation of our illustrative problem along arbitrary search direction $\boldsymbol{d}$. Subsequently, the Iris training dataset is broken into 4 equal sized mini-batches, each resulting in a unique function of $\bar{F}(\alpha)$ and $\bar{F}'(\alpha)$, plotted in green, magenta, yellow and cyan respectively.



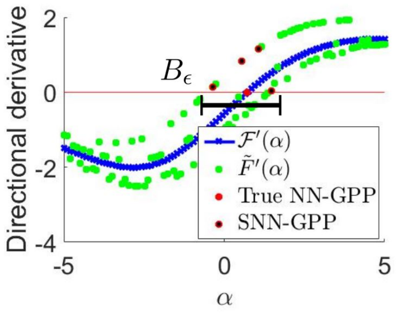
(a)



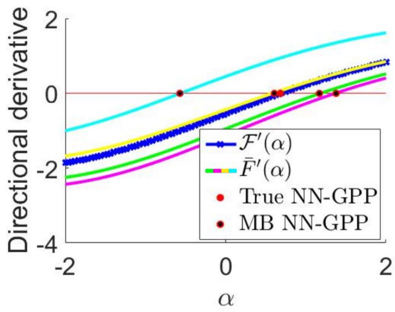
(b)



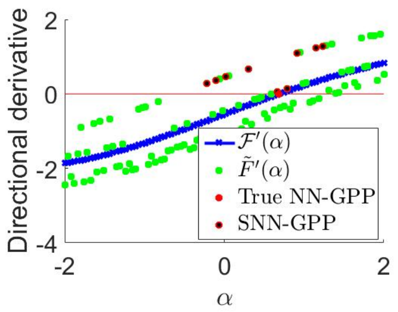
(c)



(d)



(e)



(f)

Figure 4.3: Comparing univariate loss functions and directional derivatives along a search direction with (a)(b) full-batch and static MBSS evaluated loss functions, and (c)(d) full-batch and dynamic MBSS loss functions. (e) and (f) show a closer comparison of the directional derivatives of both MBSS modes.

Both $\mathcal{F}(\alpha)$ and $\bar{F}(\alpha)$ with respective derivatives are smooth and continuous, meaning that local minima and critical points are defined. Therefore, minimization line searches are effective in finding the optima of these functions. However, note how conducting static MBSS incorporates bias into the loss approximations. Due to the nature of using a sub-set of the full training data, a *sampling error* is present for each univariate loss, resulting in the presence of a different minimizer for each mini-batch. Consequently, we define a ball B_ϵ for the MBSS case, which contains the location of all possible optima due to different mini-batches. In the 1D case, this ball simplifies to a range.

Now, suppose that dynamic MBSS is implemented in our example problem, where:

Definition 4.2.2. *Dynamic mini-batch sub-sampling is conducted when the mini-batch, $\mathcal{B}$, used to evaluate approximations $L(\boldsymbol{x})$ and $\boldsymbol{g}(\boldsymbol{x})$ changes for every evaluation, i, of $L(\boldsymbol{x})$ and $\boldsymbol{g}(\boldsymbol{x})$ in a line search along a fixed search direction for iteration, n, of a training algorithm. Therefore, mini-batches selected using dynamic MBSS are denoted as $\mathcal{B}_{n,i}$. The overhead tilde notation is used to identify approximations evaluated using dynamic MBSS as $\tilde{L}(\boldsymbol{x})$ and $\tilde{\boldsymbol{g}}(\boldsymbol{x})$ respectively. The mini-batch used to evaluate a given instance of both $\tilde{L}(\boldsymbol{x})$ and $\tilde{\boldsymbol{g}}(\boldsymbol{x})$ is identical, but subsequent evaluations of the $\tilde{L}(\boldsymbol{x})$ and $\tilde{\boldsymbol{g}}(\boldsymbol{x})$ pair prompts the re-sampling of a new mini-batch, $\mathcal{B}_{n,i}$.*

The unimodal functions constructed by applying dynamic MBSS to $F(\alpha)$ and $F'(\alpha)$ are denoted $\tilde{F}(\alpha)$ and $\tilde{F}'(\alpha)$ respectively. In our Iris example in Figure 4.3(c) and (d) we randomly select one of the four mini-batches (that result in cyan, yellow, green and magenta $\bar{F}(\alpha)$ curves) for every loss function evaluation, i. The resulting loss approximation is discontinuous in both function values and directional derivatives, as the sampling error constantly changes, depending on the mini-batch selected at the given function evaluation. This leads to local minima being identified at discontinuities over the whole sampled domain, see Figure 4.3(c). These discontinuities also imply that the first order optimality criterion $\tilde{F}'(\alpha^*) = 0$ [Arora, 2011] for a local minimum may not exist for a given instance of $\tilde{F}'(\alpha^*)$, see Figure 4.3(d), even if it may exist for the full-batch case, $\mathcal{F}(\alpha^*) = 0$.

An alternative gradient-only optimality criterion for discontinuous functions exists, namely the non-negative associated gradient projection point (NN-GPP) [Wilke et al., 2013, Snyman and Wilke, 2018] given by:

Definition 4.2.1. *NN-GPP: A non-negative associated gradient projection point (NN-GPP) is defined as any point, $\boldsymbol{x}_{nngpp}$, for which there exists $r_u > 0$ such that*

$$\nabla f(\boldsymbol{x}_{nngpp} + \lambda \boldsymbol{u})\boldsymbol{u} \geq 0, \ \ \forall \ \boldsymbol{u} \in \{\boldsymbol{y} \in \mathbb{R}^p \mid \|\boldsymbol{y}\|_2 = 1\}, \ \ \forall \ \lambda \in (0, r_u]. \tag{4.7}$$

This definition was proposed for deterministic static point-wise discontinuous functions, but generalizes to a minimum and/or a semi-definite critical point in smooth and continuous functions [Wilke et al., 2013, Snyman and Wilke, 2018]. The NN-GPP definition incorporates second order information in the form of requiring that within a certain radius, there are no descent directions away from a NN-GPP. The associated gradient [Snyman and Wilke, 2018] defines the derivative at a discontinuity. Essentially, when we only consider associated derivatives along a line search we may interpret the discontinuous function presented in Figure 4.4(a) to be equivalent to the continuous function presented in Figure 4.4(c), since both are consistent with the associated derivatives presented in Figure 4.4(b). The NN-GPP therefore filters out or ignores all discontinuities present in a discontinuous function. It is also clear, that the function minimizer of the discontinuous function, depicted as a grey dot in Figure 4.4(a), is associated with a negative directional derivative in its neighbourhood along the direction $\alpha = +1$. This implies that the global function minimizer present in the discontinuous function is not representative of a local minimum according to the associated derivatives, [Wilke et al., 2013, Snyman and Wilke, 2018]. This is because a global or local minimum would be characterized by a directional derivative going from negative to positive along a descent direction. Traditionally, for C^1 smooth functions the derivative would be zero at the local minimum, indicative of a critical point [Snyman and Wilke, 2018]. As NN-GPPs were developed specifically for discontinuous functions, [Wilke et al., 2013, Snyman and Wilke, 2018], it does not rely on the concept of a critical point as there is usually no point where the derivative is zero when discontinuous stochastic functions are considered. A NN-GPP along a search direction manifests as a sign change from negative to positive as along the descent direction. As outlined by Wilke [2012], this way of characterizing solutions of discontinuous stochastic functions is also consistent with solutions that sub-gradient algorithms or SGD (with constant step size) would find, i.e. using SGD to optimize Figure 4.4(a) would only result in converge around the NN-GPP (red dot), while the global function minimizer (grey dot) would be ignored.

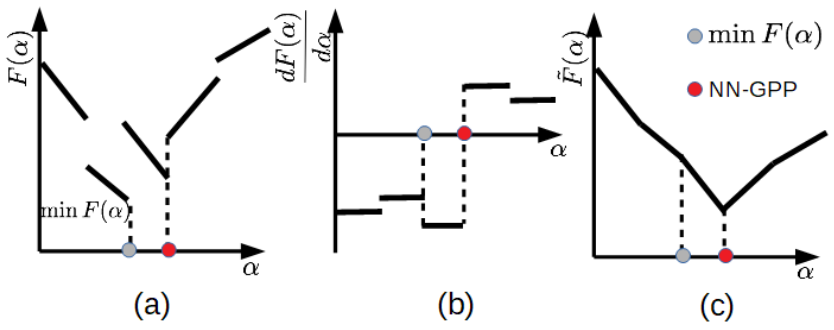


Figure 4.4: (a) Discontinuous stochastic function with (b) derivatives and (c) an alternative interpretation of (a) that is consistent with the associated derivatives given in (b). The function minimizer of $F(\alpha)$ (grey dot) and sign change from negative to positive along α (red dot) are indicated.

Returning to our Iris example in Figure 4.3(b), the NN-GPP definition correctly identifies critical points for both the true, full-batch loss function $\mathcal{F}'(\alpha)$ (True NN-GPP) and the static MBSS loss approximations $\bar{F}'(\alpha)$ (MB NN-GPP). However, jumping between batches causes sign changes to occur where there are no NN-GPPs for any individual instance of $\bar{F}'(\alpha)$, see Figure 4.3(e) and (f). Consider B_ϵ, i.e. the range between the NN-GPP of the cyan $\bar{F}'(\alpha)$, and that of the magenta $\bar{F}'(\alpha)$ curve. Alternating between instances of $\tilde{F}'(\alpha)$ in B_ϵ will result in sign changes in the sampled directional derivatives, depending on the sequence of mini-batches chosen. The locations of these sign changes are unlikely to be representative of NN-GPPs. The definition of NN-GPPs were therefore extended for this stochastic setting to stochastic NN-GPPs [Kafka and Wilke, 2019b]:

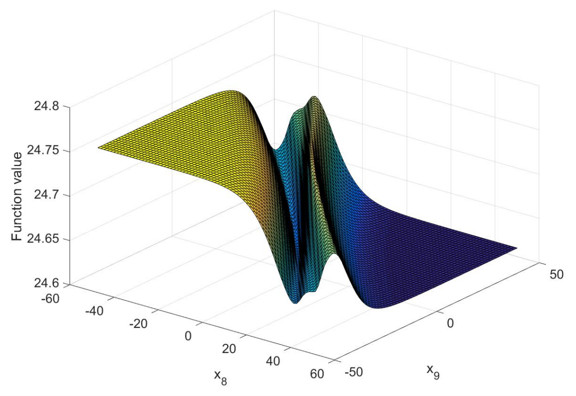
Definition 4.2.2. *SNN-GPP: A non-negative associated gradient projection point (SNN-GPP) is defined as any point, $\boldsymbol{x}_{snngpp}$, for which there exists $r_u > 0$ such that*

$$\nabla f(\boldsymbol{x}_{snngpp} + \lambda \boldsymbol{u})\boldsymbol{u} \geq 0, \ \ \forall\, \boldsymbol{u} \in \{\boldsymbol{y} \in \mathbb{R}^p \mid \|\boldsymbol{y}\|_2 = 1\}, \ \ \forall\, \lambda \in (0, r_u],$$

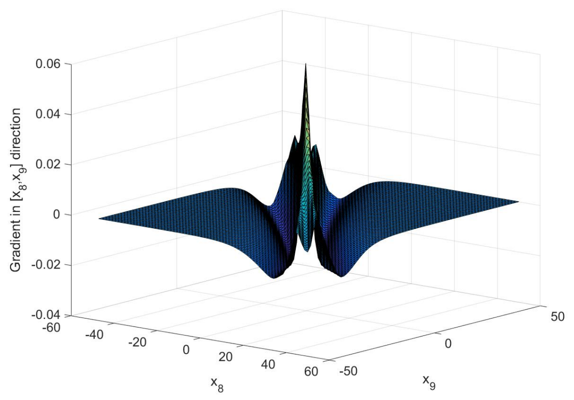
with non-zero probability.

This definition caters for dynamic MBSS, incorporating the discontinuity-filtering nature of NN-GPPs outside of B_ϵ, while accommodating sign changes in directional derivatives contained within B_ϵ that is not representative of a specific NN-GPP for any individual mini-batch. SNN-GPPs generalize to NN-GPPs, since a NN-GPP is a SNN-GPP with probability 1 for smooth and continuous functions. However, in the case of dynamic MBSS, even if $\alpha = \alpha_{nngpp}$, the probability of encountering a SNN-GPP is < 1 over a large number of sub-samples at α. This probability depends on the variance in $\tilde{\boldsymbol{g}}$, which is a function of $|\mathcal{B}_{n,i}|$ and the sampling strategy used. In our practical implementations, we limit ourselves to sampling mini-batches, $\mathcal{B}_{n,i}$, uniformly with replacement, keeping the mini-batch size, $|\mathcal{B}_{n,i}|$ constant. Subsequently, the number of possible mini-batch combinations is $K = \binom{M}{|\mathcal{B}_{n,i}|}$. When M is large, the number of combinations, K is so large, that discontinuities are often interpreted as stochastic noise [Simsekli et al., 2019]. Though this interpretation is undoubtedly useful, we suggest that strictly speaking, dynamic MBSS loss functions are discontinuous, where the magnitude of discontinuities depends on M, $|\mathcal{B}_{n,i}|$, the sampling strategy used and the data itself.

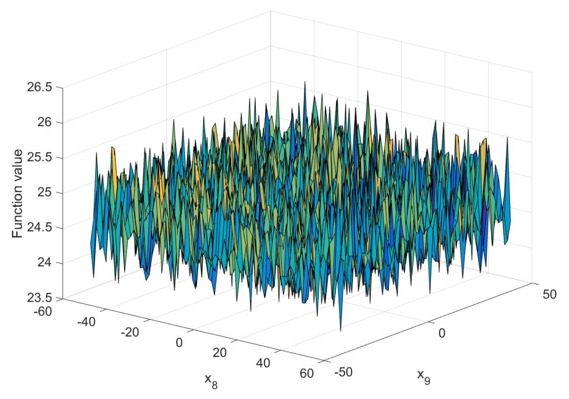
4.3 Empirical evidence of improved localization of optima with SNN-GPPs over minimizers



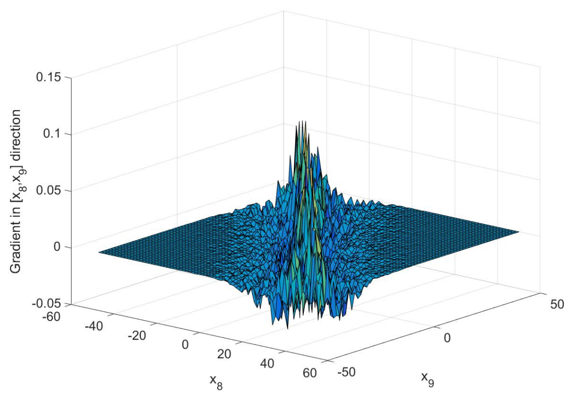
(a)



(b)



(c)



(d)

Figure 4.5: (a) Function values and (b) the directional derivatives of the loss function in dimensions x_8 and x_9 for a single hidden layer neural network applied to the Iris dataset [Fisher, 1936]. Directional derivatives are generated using a fixed search direction $\boldsymbol{d}$, where the only non-zero components are x_8 and x_9, equal to $\frac{1}{\sqrt{2}}$. The directional derivative is then evaluated as $\frac{dF_n}{d\alpha} = \boldsymbol{g}(\boldsymbol{x}_n) \cdot \boldsymbol{d}$, to generate the plots. When using full batches, both the function value and the directional derivatives are smooth and continuous functions. (c) Function values and (d) directional derivatives are discontinuous, when dynamic MBSS with mini-batch size $|\mathcal{B}_{n,i}| = 10$ is implemented. The function value plot's shape is not recognizable in comparison to (a), while directional derivatives still contain features of the original shape.

Subsequently, we present empirical evidence that SNN-GPPs offer a more representative means of identifying optima in dynamic MBSS loss functions over local minima. Firstly, we consider the nature of function value and gradient information in practical neural network loss functions with dynamic MBSS. Figure 4.6 shows weights x_8 and x_9 of our Iris network, as these happen to have interesting curvature characteristics relative to one another. In Figure 4.6(a) and (b) we show the full-batch loss and gradients in x_8 and x_9. As expected, these are smooth, continuous surfaces. As we introduce dynamic MBSS with a mini-batch size of $|\mathcal{B}_{n,i}| = 10$, sampled uniformly with replacement, the shape characteristics of the loss function are lost to the variance in the discontinuities, see Figure 4.6(a). Interestingly, the gradients are much less affected by discontinuities, particularly around the edges of the sampled domain. The centre of the domain still remains "noisy", but contrary to the function values, the shape characteristics of the dynamic MBSS gradients remain comparable to the full-batch equivalents.

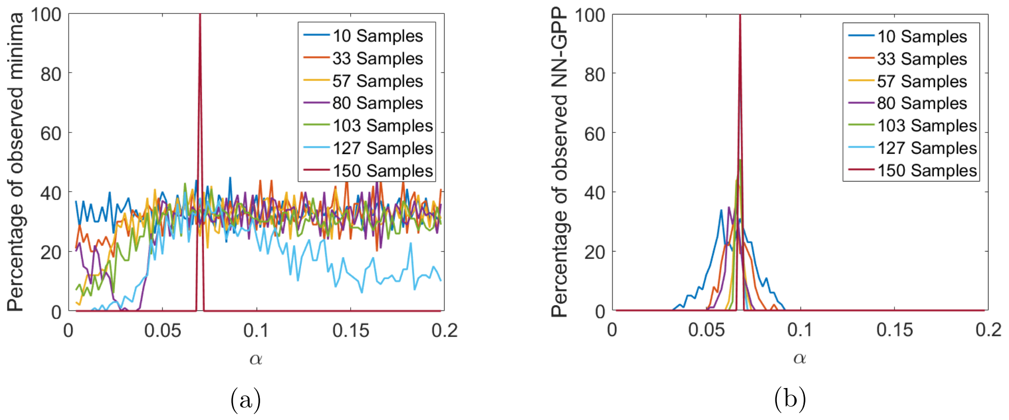


Figure 4.6: (a) Function values and (b) directional derivatives along the full-batch steepest descent direction, $\boldsymbol{d}_n = -\nabla\mathcal{L}(\boldsymbol{x})$. The loss function is obtained from the Iris classification problem of Figure 4.5 [Fisher, 1936]. The search direction is sampled by 100 points with sample sizes ranging from $|\mathcal{B}_{n,i}| = 10$ to $|\mathcal{B}_{n,i}| = M = 150$. This is repeated 100 times and the average number of minima and SNN-GPP found at every point is plotted. Minima are spread across the entire domain for most sample sizes in (a). Both minima and SNN-GPPs identify the true optimum, when the full batch is used The spatial variance of SNN-GPP is bounded around the true optimum with increasing spread for decreasing sample size. However, even with the smallest batch size, $|\mathcal{B}_{n,i}| = 10$, SNN-GPPs remain spatially bounded, unlike local minima, which are approximately uniformly spread along the sampled domain.

Now consider a hypothetical LS-SGD update performed at iteration, n, where $\boldsymbol{d}_n = -\nabla\mathcal{L}(\boldsymbol{x})$, denotes the steepest descent direction of the full-batch loss function. We note the locations of all the minimizers and SNN-GPPs along $\boldsymbol{d}_n$ over 100 increments, i, of size $\alpha_{n,i} - \alpha_{n,i-1} = 0.002$. Local minima are identified where $\tilde{F}_n(\alpha_{n,i-1}) > \tilde{F}_n(\alpha_{min}) < \tilde{F}_n(\alpha_{n,i+1})$ and SNN-GPPs where $\tilde{F}'_n(\alpha_{n,i-1}) \leq 0$ and $\tilde{F}'_n(\alpha_{snnpgpp}) > 0$. We repeat this procedure 100 times with different sample sizes $|\mathcal{B}_{n,i}|$ to approximate the likelihood of determining the locations of minima and SNN-GPPs in Figure 4.6. The spatial distribution of local minima across the sampled domain approximate a uniform distribution. The location of the true optimum is identified by full batch $|\mathcal{B}_{n,i}| = M$. Conversely, the spatial variance of SNN-GPPs are constrained in what resembles a Binomial distribution around the true optimum, with variance inversely proportional to the sample size $|\mathcal{B}_{n,i}|$. The central message of these plots is that the spatial location of SNN-GPPs is bounded within B_ϵ, making it a reliable metric to be implemented to determine step sizes in stochastic loss functions.

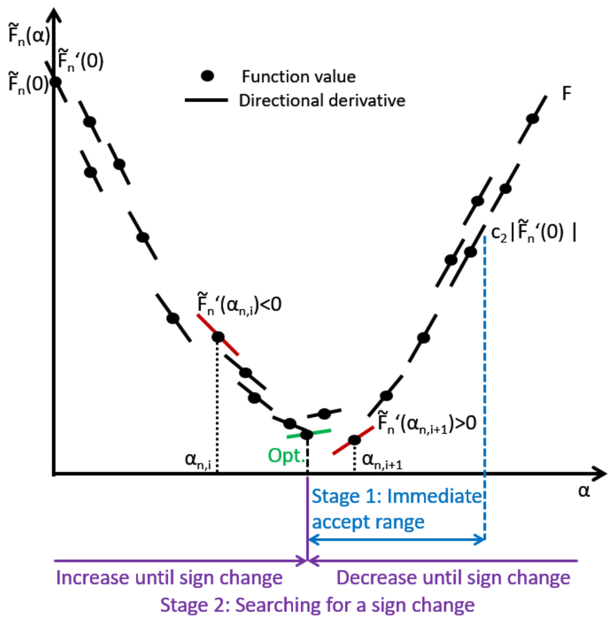
4.4 Algorithmic Details

We implement the Gradient-Only Line Search that is Inexact (GOLS-I) [Kafka and Wilke, 2019b], which requires as inputs only a given descent direction $\boldsymbol{d}_n$ and an initial step size, $\alpha_{n,0}$, which is incrementally modified. We use increment counter, i, to determine the number of modifications made to $\alpha_{n,i}$ and corresponding function evaluations performed, until a final step size α_{n,I_n} is chosen. Thus, the total number of modifications in iteration, n is I_n. Parameters which are set, but are open to modification by the user if desired are the step size scaling parameter $\eta \in \{\mathcal{R}^+ \mid \eta > 1\}$ and the modified Wolfe condition parameter $c_2 = 0.9$. For the purpose of our discussion, we use the dynamic MBSS univariate $\tilde{F}_n(\alpha)$ and $\tilde{F}'_n(\alpha)$ notation for function evaluations relating to GOLS-I. However, note that GOLS-I can be used in the context of static and dynamic MBSS as well as full-batch sampling.

The method consists of two stages: 1) Determining the adequacy of the initial guess, and 2) searching for a sign change from − to + along a descent direction. For assessing the initial guess, a modified strong Wolfe condition is implemented, to give *initial accept condition* (IAC):

$$0 < \tilde{F}_n'(\alpha_{n,0}) \leq c_2 |\tilde{F}_n'(0)|, \tag{4.8}$$

with $c_2 > 0$. If the initial guess satisfies Equation (4.8), it is immediately accepted as α_{n,I_n} and no further modification is made to the step size for the current iteration. The IAC implies that initial step size has progressed over a sign change in the directional derivative in a controlled manner, whereby the magnitude of the directional derivative is still decreased. In practice, this condition was found to be superior to the standard strong Wolfe condition, $|F_n'(\alpha)| \leq c_2 |F_n'(0)|$ [Arora, 2011].



F̃ₙ(α)
F̃ₙ'(0)
F̃ₙ(0)
● Function value
— Directional derivative
F
c₂|F̃ₙ'(0) |
F̃ₙ'(αₙ,ᵢ)<0
F̃ₙ'(αₙ,ᵢ₊₁)>0
αₙ,ᵢ
Opt.
αₙ,ᵢ₊₁
α
Stage 1: Immediate accept range
Increase until sign change
Decrease until sign change
Stage 2: Searching for a sign change

Figure 4.7: Schematic diagram of the Gradient-Only Line Search that is Inexact (GOLS-I): Dynamic MBSS results in discontinuous loss functions. $\tilde{F}'(\alpha_{n,0})$ is tested on the initial accept condition, Equation (4.8), if this holds, $\alpha_{n,0}$ is accepted. Otherwise, the directional derivative sign of $\tilde{F}'(\alpha_{n,0})$ determines whether the step size needs to be increased by Equation (4.9) or decreased by Equation (4.10) until an SNN-GPP is isolated.

If the initial accept condition is not satisfied, the algorithm enters stage 2, where the initial step size is increased or decreased by factor η until a sign change is observed, i.e.

- If $\tilde{F}_n'(\alpha_{n,0}) < 0$, then
$$\alpha_{n,i+1} = \eta \alpha_{n,i}, \tag{4.9}$$
with $i := i + 1$ until $\tilde{F}_n'(\alpha_{n,i}) > 0$, or
- if $\tilde{F}_n'(\alpha_{n,0}) > 0$, then
$$\alpha_{n,i+1} = \frac{\alpha_{n,i}}{\eta}, \tag{4.10}$$
with $i := i + 1$ until $\tilde{F}_n'(\alpha_{n,i}) < 0$.

This process is illustrated in Figure 4.7. An upper and lower limit is given for step sizes to ensure algorithmic stability in cases of monotonically decreasing or ascending search directions respectively. The maximum allowable step size is inspired by convergent fixed step sizes according to the Lipschitz condition [Boyd and Park, 2014]. We therefore choose the maximum step size conservatively as:

$$\alpha_{max} = \min(\frac{1}{\|\boldsymbol{d}_n\|_2}, 10^7). \tag{4.11}$$

Conservative updates are enforced by ensuring that $\alpha < \frac{1}{\|\boldsymbol{d}_n\|_2}$, which restricts the step size in the case of steep descent directions, but allows larger step sizes for more gradual descent directions. The absolute upper limit restricts overly large step sizes in the case of flat search directions. Taking the minimum of the two limits ensures that the line search can traverse both steep declines and flat planes, while remaining stable in the case of spurious mini-batch characteristics.

The minimum step size avoids high computational cost, should the line search encounter an ascent direction. In such cases computational resources are wasted by continually decreasing the step size towards zero. Therefore the minimum step size is limited to:

$$\alpha_{min} = 10^{-8}, \tag{4.12}$$

which in combination with the maximum step size results in an available step size range of 15 orders of magnitude. Unlike PrLS [Mahsereci and Hennig, 2017], GOLS-I does not limit the number of function evaluations per iteration, which makes the entire range of step size magnitudes available to the line search in pursuit of a directional derivative sign change from $-$ to $+$ along a descent direction.

In this study, the first iteration of GOLS-I ($n = 0$), the initial guess is selected to be $\alpha_{0,0} = \alpha_{min}$. Therefore, GOLS-I is tasked with growing the step size from α_{min} to the desired magnitude as dictated by the presented loss function and descent direction. In subsequent iterations, the initial guess is set to be the final step size of the previous iteration, i.e. $\alpha_{n,0} = \alpha_{n-1,I_n}$. The pseudo code for GOLS-I is given in Algorithm 2.

Algorithm 2: GOLS-I: Gradient-Only Line Search that is Inexact

```
   Input: $\tilde{F}'_n(\alpha)$, $d_n$ , $\alpha_{n,0}$
   Output: $\alpha_{n,I_n}$, $I_n$
1  Define constants: $\alpha_{min} = 10^{-8}$, flag = 1, $\eta = 2$, $c_2 = 0.9$, $i = 0$
2  $\alpha_{max} = min(\frac{1}{||d_n||_2}, 10^7)$
3  Evaluate $\tilde{F}'_n(0)$, increment $i$ (or use saved gradient from last $F'_{n-1}(\alpha_{n-1,I_{n-1}})$, to
     evaluate $\tilde{g}(x_{n-1} + \alpha_{n-1,I_{n-1}} \cdot d_{n-1})^T d_n$ without incrementing $i$)
4  if $\alpha_{n,0} > \alpha_{max}$ then
5      $\alpha_{n,0} = \alpha_{max}$
6  if $\alpha_{n,0} < \alpha_{min}$ then
7      $\alpha_{n,0} = \alpha_{min}$
8  Evaluate $\tilde{F}'_n(\alpha_{n,0})$, increment $i$
9  Define $tol_{dd} = |c_2 \tilde{F}'_n(0)|$
10 if $\tilde{F}'_n(\alpha_{n,0}) > 0$ and $\alpha_{n,0} < \alpha_{max}$ then
11     flag = 1, decrease step size
12 if $\tilde{F}'_n(\alpha_{n,0}) < 0$ and $\alpha_{n,0} > \alpha_{min}$ then
13     flag = 2, increase step size
14 if $\tilde{F}'_n(\alpha_{n,0}) > 0$ and $\tilde{F}'_n(\alpha_{n,0}) < tol_{dd}$ then
15     flag = 0, immediate accept condition
16 while flag > 0 do
17     if flag = 2 then
18         $\alpha_{n,i+1} = \alpha_{n,i} \cdot \eta$
19         Evaluate $\tilde{F}'_n(\alpha_{n,i+1})$
20         if $\tilde{F}'_n(\alpha_{n,i+1}) \geq 0$ then
21             flag = 0
22         if $\alpha_{n,i+1} > \frac{\alpha_{max}}{\eta}$ then
23             flag = 0
24     if flag = 1 then
25         $\alpha_{n,i+1} = \frac{\alpha_{n,i}}{\eta}$
26         Evaluate $\tilde{F}'_n(\alpha_{n,i+1})$
27         if $\tilde{F}'_n(\alpha_{n,i+1}) < 0$ then
28             flag = 0
29         if $\alpha_{n,i+1} < \alpha_{min} \cdot \eta$ then
30             flag = 0
31 $\alpha_{n,I_n} = \alpha_{n,i+1}$
```

4.4.1 Proof of Global Convergence for Full-Batch Sampling

Concerning notation for the following proofs, the input variable $\boldsymbol{x}$ for loss functions is omitted in aid of brevity, such that $\mathcal{L}(\boldsymbol{x})$ is represented simply as $\mathcal{L}$. Therefore, suppose that the loss function $\mathcal{L}$ obtained from full batch sampling is smooth, coercive with a unique minimizer $\boldsymbol{x}^*$. Any Lipschitz function $\hat{\mathcal{L}}$ can be regularized to be coercive using Tikhonov regularization with a sufficient large regularization coefficient.

The iteration updates of an optimization algorithm can be considered as a dynamical system in discrete time:

$$\boldsymbol{x}_{n+1} = \mathcal{D}(\boldsymbol{x}_n),\ \mathcal{D} : \mathcal{R}^p \rightarrow \mathcal{R}^p. \tag{4.13}$$

It follows from Lyapunov's global stability theorem [Lyapunov, 1992] in discrete time that any Lyapunov function $\Gamma(\boldsymbol{x})$ defined by positivity, coercive and strict decrease:

1. Positivity: $\Gamma(\mathbf{0}) = \mathbf{0}$ and $\Gamma(\boldsymbol{x}) > 0$, $\forall \boldsymbol{x} \neq \mathbf{0}$
2. Coercive: $\Gamma(\boldsymbol{x}) \rightarrow \infty$ as $\boldsymbol{x} \rightarrow \infty$
3. Strict descent: $\Gamma(\mathcal{D}(\boldsymbol{x})) < \Gamma(\boldsymbol{x})$, $\forall\ \boldsymbol{x} \neq \mathbf{0}$,

results in $\boldsymbol{x}_n \rightarrow \mathbf{0}$ as $n \rightarrow \infty$, $\forall\ \boldsymbol{x}_0 \in \mathcal{R}^p$.

Theorem 4.4.1. *Let $f(\boldsymbol{x})$ be any smooth coercive function with a unique global minimum $\boldsymbol{x}^*$, for $\boldsymbol{x}_{n+1} = \mathcal{D}(\boldsymbol{x}_n)$, $\forall\ \boldsymbol{x}_n \neq \boldsymbol{x}_n^*$ restricted such that $f(\beta \boldsymbol{x}_{n+1} + (1-\beta)\boldsymbol{x}_n) < f(\boldsymbol{x}_n)$, $\forall\ \beta \in (0, 1]$. Then $\mathcal{D}$ will result in updates that are globally convergent.*

Let the error at iteration n be given by $\boldsymbol{e}_n := \boldsymbol{x}_n - \boldsymbol{x}^*$ for which we can construct the Lyapunov function $\Gamma(\boldsymbol{e}) = f(\boldsymbol{e} + \boldsymbol{x}^*) - f(\boldsymbol{x}^*)$. It follows that $\Gamma(\mathbf{0}) = \mathbf{0}$ and that $\Gamma(\boldsymbol{e}) > 0$, $\forall\ \boldsymbol{e} \neq \mathbf{0}$, since $\boldsymbol{x}^*$ is a unique global minimum of f.

At every iteration our line search update locates an SNN-GPP along the descent direction $\boldsymbol{d}_n$, by locating a sign change from negative to positive along $\boldsymbol{d}_n$. Wilke et al. [2013] proved this to be equivalent to minimizing along $\boldsymbol{d}_n$ when $f(\boldsymbol{x}_n + \alpha \boldsymbol{d}_n)$ is smooth and the sign of the directional derivative $\nabla^{\mathrm{T}} f(\boldsymbol{x}_n + \alpha \boldsymbol{d}_n)\boldsymbol{d}_n$, is negative $\forall\ \alpha \in [0, \alpha_n^*)$ along $\boldsymbol{d}_n$. Here, α_n^* defines the step size to the first optimum along the search direction $\boldsymbol{d}_n$. It is therefore guaranteed that $f(\boldsymbol{x}_{n+1}) < f(\boldsymbol{x}_n)$ at every iteration n. In addition, $f(\beta \boldsymbol{x}_{n+1} + (1-\beta)\boldsymbol{x}_n) < f(\boldsymbol{x}_n)$, $\forall\ \beta \in (0, 1]$ ensures that for our choice of discrete dynamical update $\mathcal{D}$, we can always make progress unless $\boldsymbol{x}_n = \boldsymbol{x}^*$. Hence, for any $\boldsymbol{e}_n \neq \mathbf{0}$ it follows that

$$\Gamma(\boldsymbol{e}_{n+1}) = f(\boldsymbol{x}_{n+1} - \boldsymbol{x}^* + \boldsymbol{x}^*) - f(\boldsymbol{x}^*) < f(\boldsymbol{x}_n - \boldsymbol{x}^* + \boldsymbol{x}^*) - f(\boldsymbol{x}^*) = \Gamma(\boldsymbol{e}_n).$$

It then follows from Lyaponov's global stability theorem that $\boldsymbol{e}_n \rightarrow \mathbf{0}$ as $n \rightarrow \infty$. Hence $\forall\ \boldsymbol{x}_0$ we have that $\boldsymbol{x}_n \rightarrow \boldsymbol{x}^*$, which proves that finding an SNN-GPP at every iteration n results in a globally convergent strategy.

4.4.2 Proof of Global Convergence for Dynamic Mini-Batch Sub-Sampling

Consider the discontinuous loss function $\tilde{L}$ obtained through dynamic mini-batch sub-sampling with smooth expected response $E[\tilde{L}]$ and unique expected minimizer $\boldsymbol{x}^*$. Assume that the function $\tilde{L}$ is directional derivative coercive (see Wilke et al. [2013]) around a ball $\boldsymbol{x} \in \hat{B}_\epsilon(\boldsymbol{x}) = \{\boldsymbol{q} \mid \|\boldsymbol{q} - \boldsymbol{x}^*\| \geq \epsilon\}$ of given radius $\epsilon \in \mathcal{R} > 0$ that is centred around the expected minimizer $\boldsymbol{x}^*$. This implies that for given radius ϵ and for any point outside the ball $\boldsymbol{x}_1 \in \hat{B}_\epsilon(\boldsymbol{x}_1)$ and any point inside the ball $\boldsymbol{x}_2 \in B_\epsilon(\boldsymbol{x}_2) = \{\boldsymbol{q} \mid \|\boldsymbol{q} - \boldsymbol{x}^*\| < \epsilon\}$ with $\boldsymbol{u} = \boldsymbol{x}_1 - \boldsymbol{x}_2$ the following must hold:

$$\nabla f(\boldsymbol{x}_1)^T \boldsymbol{u} > 0. \tag{4.14}$$

As before, the iteration updates of an optimization algorithm can be considered as a dynamical system in discrete time:

$$\boldsymbol{x}_{n+1} = \mathcal{D}(\boldsymbol{x}_n),\ \mathcal{D} : \mathcal{R}^p \rightarrow \mathcal{R}^p. \tag{4.15}$$

We relax Lyapunov's global stability theorem in discrete time for dynamic MBSS discontinuous functions, such that any smooth expected Lyapunov function $E[\Gamma(\boldsymbol{x})]$ defined by expected positivity, coercive and expected strict decrease around a ball $\boldsymbol{x} \in \hat{B}_\epsilon(\boldsymbol{x}) = \{\boldsymbol{q} \mid \|\boldsymbol{q} - \boldsymbol{x}^*\| \geq \epsilon\}$ of given radius $\epsilon \in \mathcal{R} > 0$, with:

1. Expected positivity: $E[\Gamma(\mathbf{0})] = \mathbf{0}$ and $E[\Gamma(\boldsymbol{x})] > 0$, $\forall\ \boldsymbol{x} \neq \mathbf{0}$
2. Coercive: $\Gamma(\boldsymbol{x}) \rightarrow \infty$ as $\boldsymbol{x} \rightarrow \infty$
3. Directional derivative coercive for any point $\boldsymbol{x} \in \hat{B}_\epsilon(\boldsymbol{x}) = \{\boldsymbol{q} \mid \|\boldsymbol{q} - \boldsymbol{x}^*\| > \epsilon\}$ of radius $\epsilon \in \mathcal{R} > 0$

4. Expected strict descent: $E[\Gamma(\mathcal{D}(\boldsymbol{x}))] < E[\Gamma(\boldsymbol{x})]$, $\forall\ \boldsymbol{x} \neq \mathbf{0}$,

results in $\boldsymbol{x}_n \in B_\epsilon(\boldsymbol{x}_n) = \{\boldsymbol{q} \mid \|\boldsymbol{q} - \boldsymbol{x}^*\| < \epsilon\}$ as $n \to \infty$, $\forall\ \boldsymbol{x}_0 \in \mathcal{R}^p$.

Theorem 4.4.2. *Let $f(\boldsymbol{x})$ be any smooth expected coercive function with a unique expected global minimum $\boldsymbol{x}^*$ that is directional derivative coercive around a ball $\hat{B}_\epsilon(\boldsymbol{x}) = \{\boldsymbol{q} \mid \|\boldsymbol{q} - \boldsymbol{x}^*\| \geq \epsilon\}$ of radius $\epsilon \in \mathcal{R} > 0$. Then $\boldsymbol{x}_{n+1} = \mathcal{D}(\boldsymbol{x}_n)$, $\forall\ \boldsymbol{x}_n \in \hat{B}_\epsilon(\boldsymbol{x}_n)$ is restricted such that $\nabla^T f(\boldsymbol{x}_n + \alpha\boldsymbol{d}_n)\boldsymbol{d}_n < 0$, $\forall\ \alpha \in [0, \alpha_n^*)$ along descent direction $\boldsymbol{d}_n$. Then $\mathcal{D}$ will result in updates that globally converge to the ball $B_\epsilon(\boldsymbol{x}) = \{\boldsymbol{q} \mid \|\boldsymbol{q} - \boldsymbol{x}^*\| < \epsilon\}$ of radius $\epsilon \in \mathcal{R} > 0$ centred around $\boldsymbol{x}^*$.*

Let the error at iteration n be given by $\boldsymbol{e}_n := \boldsymbol{x}_n - \boldsymbol{x}^*$ for which we can construct the Lyapunov function $\Gamma(\boldsymbol{e}) = f(\boldsymbol{e} + \boldsymbol{x}^*) - f(\boldsymbol{x}^*)$ and expected Lyapunov function $E[\Gamma(\boldsymbol{e})] = E[f(\boldsymbol{e} + \boldsymbol{x}^*)] - E[f(\boldsymbol{x}^*)]$. It follows that $E[\Gamma(\mathbf{0})] = \mathbf{0}$ and that $E[\Gamma(\boldsymbol{e})] > 0$, $\forall\ \boldsymbol{e} \neq \mathbf{0}$, since $\boldsymbol{x}^*$ is a unique expected global minimum of f.

At every iteration our line search update locates an SNN-GPP along the descent direction $\boldsymbol{d}_n$, by locating a sign change from negative to positive along $\boldsymbol{d}_n$. Since the function is smooth expected coercive and directional derivative coercive around a ball $\hat{B}_\epsilon(\boldsymbol{x})$, expected descent follows $E[\Gamma(\mathcal{D}(\boldsymbol{x}))] < E[\Gamma(\boldsymbol{x})]$, $\forall\ \boldsymbol{x} \notin B_\epsilon(\boldsymbol{x}) = \{\boldsymbol{q} \mid \|\boldsymbol{q} - \boldsymbol{x}^*\| < \epsilon\}$ of radius $\epsilon \in \mathcal{R} > 0$. It is therefore guaranteed that $E[f(\boldsymbol{x}_{n+1})] < E[f(\boldsymbol{x}_n)]$ at every iteration n. In addition, $E[f(\beta\boldsymbol{x}_{n+1} + (1-\beta)\boldsymbol{x}_n)] < E[f(\boldsymbol{x}_n)]$, $\forall\ \beta \in (0, 1]$ ensures that for our choice of discrete dynamical update $\mathcal{D}$, we can always make progress unless $\boldsymbol{x}_n \in B_\epsilon(\boldsymbol{x}) = \{\boldsymbol{q} \mid \|\boldsymbol{q} - \boldsymbol{x}^*\| < \epsilon\}$. In addition, since the function is directional derivative coercive around the ball $B_\epsilon(\boldsymbol{x}_n)$, any point $\boldsymbol{x} \in B_\epsilon(\boldsymbol{x}_n)$ remains in $B_\epsilon(\boldsymbol{x}_n)$ due to the update requirement of a sign change from negative to positive along the descent direction. Hence, for any $\boldsymbol{e}_n$ such that $\|\boldsymbol{e}_n\| > \epsilon$ it follows that:

$$E[\Gamma(\boldsymbol{e}_{n+1})] = E[f(\boldsymbol{x}_{n+1} - \boldsymbol{x}^* + \boldsymbol{x}^*)] - E[f(\boldsymbol{x}^*)] < E[f(\boldsymbol{x}_n - \boldsymbol{x}^* + \boldsymbol{x}^*)] - E[f(\boldsymbol{x}^*)] = E[\Gamma(\boldsymbol{e}_n)].$$

It then follows from Lyaponov's relaxed global stability theorem that $\boldsymbol{e}_n \in B_\epsilon(\boldsymbol{x})$ as $n \to \infty$. Hence $\forall\ \boldsymbol{x}_0$ we have that $\boldsymbol{x}_n \in \hat{B}_\epsilon(\boldsymbol{x}_n) \to \boldsymbol{x}_n \in B_\epsilon(\boldsymbol{x}_n)$ as $n \to \infty$, which proves that finding an SNN-GPP at every iteration n results in a globally converges to the ball $B_\epsilon(\boldsymbol{x}_n)$.

4.5 Numerical Studies

The network architectures and problems implemented in the numerical investigations of this chapter are predominantly taken from the work of [Mahsereci and Hennig, 2017], which serves as the benchmark against which we compare GOLS-I. The work by Mahsereci and Hennig [2017] introduces PrLS, which is used to estimate step sizes in stochastic gradient descent [Robbins and Monro, 1951] with a line search (LS-SGD) and compared to standard SGD with *a priori* fixed step sizes. We directly compare the performances of PrLS and GOLS-I for determining step sizes for LS-SGD using the Matlab code for PrLS supplied[1] in the relevant article [Mahsereci and Hennig, 2017]. The datasets we consider are:

- Breast Cancer Wisconsin Diagnostic (BCWD) Dataset [Street et al., 1993], a binary classification problem, distinguishing between "benign" and "malignant" tumours, using 30 different features;
- MNIST Dataset [Lecun et al., 1998], a multi-class classification problem with images of handwritten digits from 0 to 9 in grey-scale with a resolution of 28x28 pixels; and
- CIFAR10 [Krizhevsky and Hinton, 2009], a multi-class classification problem with images of 10 natural objects such as deer, cats, dogs, ships, etc. with colour images of resolution 32x32.

[1]https://ei.is.tuebingen.mpg.de/publications/mahhen2015

Table 4.1 gives further details about the datasets, as well as parameters specific to their implementations such as: The neural network architectures used, number of function evaluations to which training is limited and mini-batch sizes used. These details are implemented as prescribed by Mahsereci and Hennig [2017]. GOLS-I was implemented in both Matlab and PyTorch 1.0. The Matlab implementations are used for direct comparisons between GOLS-I and PrLS, while the PyTorch implementation demonstrates how GOLS-I compares to fixed step sizes. This latter implementation also demonstrates the ease by which GOLS-I is transferred to GPU-capable computing platforms. All datasets were pre-processed using the standard transformation (Z-score) for each input dimension in the dataset.

Datset	**Training obs.**	**Test obs.**	**Input dim.**	**Output dim.**	**Net structure**	**Max. F.E.**	$\vert\mathcal{B}_{n,i}\vert$ **in training**
BCWD	400	169	30	2	Log. Regression, NetPI, NetPII	3000/ 100000	10,50, 100,400
MNIST	50000	10000	784	10	NetI, NetII	40000	10,100, 200
CIFAR	10000 (Batch1)	10000	3072	10	NetI, NetII	10000	10,100, 200

Table 4.1: Relevant parameters related to the datasets used in numerical experiments.

Following Mahsereci and Hennig [2017], both MNIST and CIFAR10 are implemented using two different network architectures, NetI and NetII, while the smaller BCWD dataset was trained using logistic regression and two single hidden layer fully connected networks, NetPI and NetPII. The latter two network architectures are borrowed from the work of Prechelt [1994], in which a modified version of the BCWD dataset was implemented. The parameters governing the different implementations are summarized in Table 4.2. This constitutes a total of 7 combinations of different datasets, architectures and loss functions used in the numerical study. All networks are fully connected, and although the detail given concerning the hidden layers of the network excludes the biases in Table 4.2, they are included in the implementations. Mahsereci and Hennig [2017] have stated that a normal distribution was used to initialize all networks. However, we found that the problems using NetII would not converge using normally distributed weight initializations unless the variance was reduced to 0.1 for MNIST and 0.01 for CIFAR10 respectively. The latter variance scaling was also adopted for the NetI implementations of CIFAR10. For each problem, 10 training runs were conducted, where training and test classification errors evaluated, as well as estimated step sizes are noted during training. The resolution of the classification error plots in the log domain is limited by the size of the respective training and test datasets. For the BCWD dataset, this is the full test dataset, while for MNIST and CIFAR10, the classification errors are evaluated for a random subsample of 1000 observations from the training and test datasets respectively. Therefore, to avoid bad scaling in error plots for cases where the classification error is 0, a numerical constant of $1 \cdot 10^{-4}$ was added to classification error calculations. Therefore the value $1 \cdot 10^{-4}$ on classification error plots represents absolute zero.

For our PyTorch implementation of the BCWD logistic regression problem, we selected three constant step sizes, each one order of magnitude apart, $\alpha_{n,I_n} \in \{1, 10, 100\}$, ensuring that the full training performance modality is captured. This means that the small fixed step size represents a slow and overly conservative learning rate that leads to wasted gradient computations during training. The medium fixed step results in an effective and efficient learning rate with desired convergence performance and the large fixed step in training that is too aggressive and usually leads to detrimental performance. GOLS-I's training performance is compared to the three constant learning rates, to investigate its feasibility in replacing *a priori* determined fixed step sizes.

There is a significant difference in the information required by PrLS and GOLS-I. Apart from using function value and gradient information, which is readily available in most neural network software, such as Tensorflow [tensorflow.org, 2019] and PyTorch [pytorch.org, 2019],

Network	Hidden layer structure	Activation function	Initialization	Loss function
Logistic regression	N/A	Sigmoid	$\mathcal{N}(0, I)$	BCE
NetPI	32	Sigmoid	$\mathcal{N}(0, I)$	BCE
NetPII	32	Sigmoid	$\mathcal{N}(0, I)$	MSE
NetI	800	Sigmoid	$\mathcal{N}(0, I)$ MNIST $\mathcal{N}(0, 0.01I)$ CIFAR	BCE
NetII	1000,500,250	Tanh	$\mathcal{N}(0, 0.1I)$ MNIST $\mathcal{N}(0, 0.01I)$ CIFAR	MSE

Table 4.2: Parameters and settings governing the implemented network architectures and their training.

PrLS also requires variance estimates of both function values and gradients, which are not standard outputs. Conversely, GOLS-I requires only gradients to be evaluated, in order to calculate directional derivatives. However, since gradients are evaluated via backpropagation, function values are evaluated as a by-product. In order to be consistent with the information presented to PrLS and GOLS-I in our direct comparisons (using Matlab), we consider a function evaluation to be: A function value, the respective gradient, as well as the in-batch variance estimates of function value and gradients as prescribed by Mahsereci and Hennig [2017]. Of this information, GOLS-I then only uses the gradient vector, while PrLS uses all of the above. In our PyTorch implementation of GOLS-I, only the gradient vector is evaluated.

Both PrLS and GOLS-I require an initial guess at the beginning of a training run, after which the step size of the previous iteration is used as the initial guess for the next iteration, i.e. $\alpha_{n,0} = \alpha_{n-1,I_n}$. The starting initial guess for PrLS is set as $\alpha_{0,0} = 10^{-4}$, as prescribed by Mahsereci and Hennig [2017], while GOLS-I uses a more conservative $\alpha_{0,0} = 10^{-8}$, as discussed in Section 4.4. The minimum number of function evaluations per iteration is 1 for both PrLS and GOLS-I, when the initial condition is satisfied. Conversely, the maximum number of function evaluations for PrLS is capped at 7, while GOLS-I is uncapped in function evaluations, but practically capped by the range of available step sizes until α_{max} is reached.

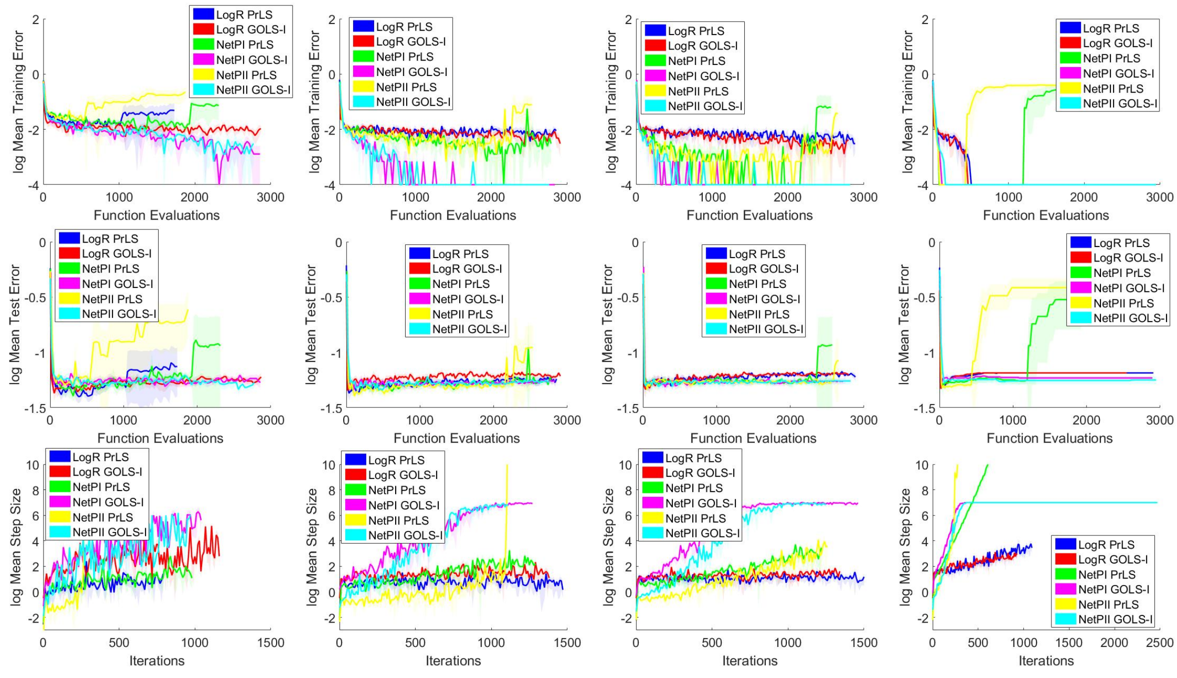
4.6 Results

Results are categorized according to the datasets and respective architectures investigated, where the training and test classification errors, as well as estimated step sizes are shown for different mini-batch sizes, $|\mathcal{B}_{n,i}|$. Note, that classification error plots are shown in terms of function evaluations, while step sizes are shown in terms of iterations. This allows the training performances between PrLS and GOLS-I to be compared in terms of computational cost, while also showing the difference in function evaluations per iteration in step size plots.

4.6.1 The Breast Cancer Wisconsin Diagnostic (BCWD) Dataset with Logistic Regression, NetPI and NetPII

Training and test classification error as well as corresponding step sizes for the different network architectures used with the BCWD dataset are shown in Figure 4.8. Note, that in order to clearly distinguish and compare the performances of the investigated architectures for this problem, we plot the mean over 10 runs using a solid line, while the variance cloud is indicated by a shaded area around the mean. This representation is not used for subsequent figures, where each of the 10 runs is plotted individually, as determined by Mahsereci and Hennig [2017].

In the logistic regression (LogR) example, the performance of GOLS-I and PrLS is similar for mini-batch sizes of $|\mathcal{B}_{n,i}| \geq 50$. In these cases, the step sizes estimated by both line searches are also of comparable magnitude. This shows that both PrLS and GOLS-I automatically adapt their starting initial guesses from $\alpha_{0,0} = 1 \cdot 10^{-4}$ and $\alpha_{0,0} = 1 \cdot 10^{-8}$ respectively to magnitudes around 1 and larger. For the logistic regression problem we observe an increase in step sizes as a function of iterations for both methods. On average, the number of gradient evaluations per iteration is in the low 2s for the BCWD dataset, but can be up to 17 for individual iterations of GOLS-I. However, on average we see that GOLS-I and PrLS also complete a similar number of iterations for a fixed maximum number of function evaluations. This suggests, that the performance between the two line searches is competitive, an assertion supported by the almost indistinguishable training, test and step size plots for the full batch case, $M = 400$.



(a) $|\mathcal{B}_{n,i}| = 10$ (b) $|\mathcal{B}_{n,i}| = 50$ (c) $|\mathcal{B}_{n,i}| = 100$ (d) $M = 400$

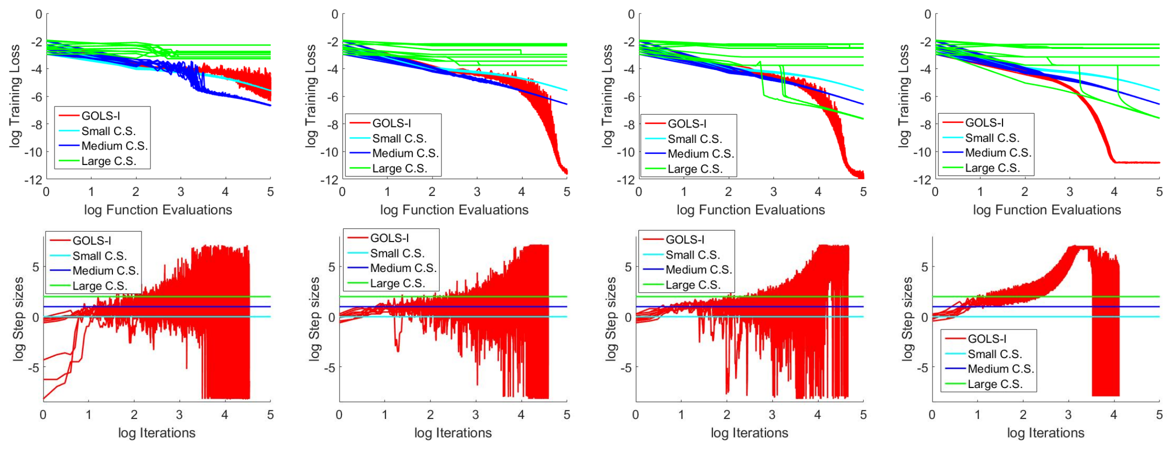
Figure 4.8: Log training error, log training loss, log test error and the log of the step sizes as obtained with various batch sizes for the BCWD dataset problem.

However, for $|\mathcal{B}_{n,i}| = 10$ PrLS exhibits unstable behaviour, encountering numerical difficulties that resulted in early termination. This is in contrast to the results shown by Mahsereci and Hennig [2017], which suggest stability with $|\mathcal{B}_{n,i}| = 10$ for 10,000 function evaluations. We observe this unstable behaviour for all three architectures trained with PrLS and $|\mathcal{B}_{n,i}| = 10$, while conversely, GOLS-I completed all training runs with $|\mathcal{B}_{n,i}| = 10$.

Next, consider the results of NetPI and NetPII with $|\mathcal{B}_{n,i}| \geq 50$. Training performance of both line searches improves as a function of increasing mini-batch size. However, the performance difference between GOLS-I and PrLS becomes more distinct for these network architectures. The use of GOLS-I produces instances where the training classification errors are 0. In such cases, the lower bound of the y-axis is limited to the numerical nugget of 10^{-4}. The smallest non-zero error for BCWD is $log(\frac{1}{400}) \approx -2.6$. For GOLS-I with $|\mathcal{B}_{n,i}| = 50$, most training runs reach zero training classification error after approximately 1000 function evaluations. This is accelerated to 500-800 function evaluations for $|\mathcal{B}_{n,i}| = 100$. The use of PrLS does not produce perfect training results for $|\mathcal{B}_{n,i}| = 50$, but does improve for $|\mathcal{B}_{n,i}| = 100$, first reaching 10^{-4} after ±1200 function evaluations. This difference in performance is due to the difference in step sizes determined by GOLS-I and PrLS for $|\mathcal{B}_{n,i}| \in \{50, 100\}$. GOLS-I grows its step size at a significantly faster rate than PrLS for both NetPI and NetPII, reaching α_{max} while still remaining stable. This means that the step sizes determined by PrLS for $|\mathcal{B}_{n,i}| \in \{50, 100\}$ are more conservative. The step size behaviour of PrLS changes significantly for $M = 400$, where the step sizes of both methods grow at a similar rate until ±400 function evaluations, after which GOLS-I reaches its maximum step size, but PrLS continues to grow its steps. This has the consequence that PrLS becomes unstable after latest 1200 function evaluations. We postulate, that this weakness of PrLS is related to cases where the norm of the gradient vector approaches 0 towards the end of training, an occurrence which we observe in training and test errors for all mini-batch sizes.

Apart from the late-training divergence of PrLS, the test classification errors between line search methods and network architectures are largely the same for this problem. Overfitting occurs readily after around 200 function evaluations, with the logistic regression implementations showing higher test classification errors in later training than both NetPI and NetPII. This indicates, that both PrLS and GOLS-I are in principle capable of fitting the given architectures to the BCWD dataset. However, for this example GOLS-I has improved training performance to PrLS, as well as exhibiting superior algorithmic stability.

Figure 4.9 shows log training losses and step sizes for the comparison between GOLS-I and fixed step sizes applied to the logistic regression problem with the BCWD dataset over 100, 000 function evaluations, as first demonstrated by Mahsereci and Hennig [2017]. In these plots we show all 10 runs for GOLS-I and each constant step size. In terms of constant step sizes, the small step size exhibits slow convergence, the medium step size performs well, and the large step size often leads to divergence. As the batch size increases, the performance of the large constant step size becomes competitive for isolated instances, as observed for $|\mathcal{B}_{n,i}| = 100$ and $|\mathcal{B}_{n,i}| = 400$. However, overall the large step size has the largest variance.



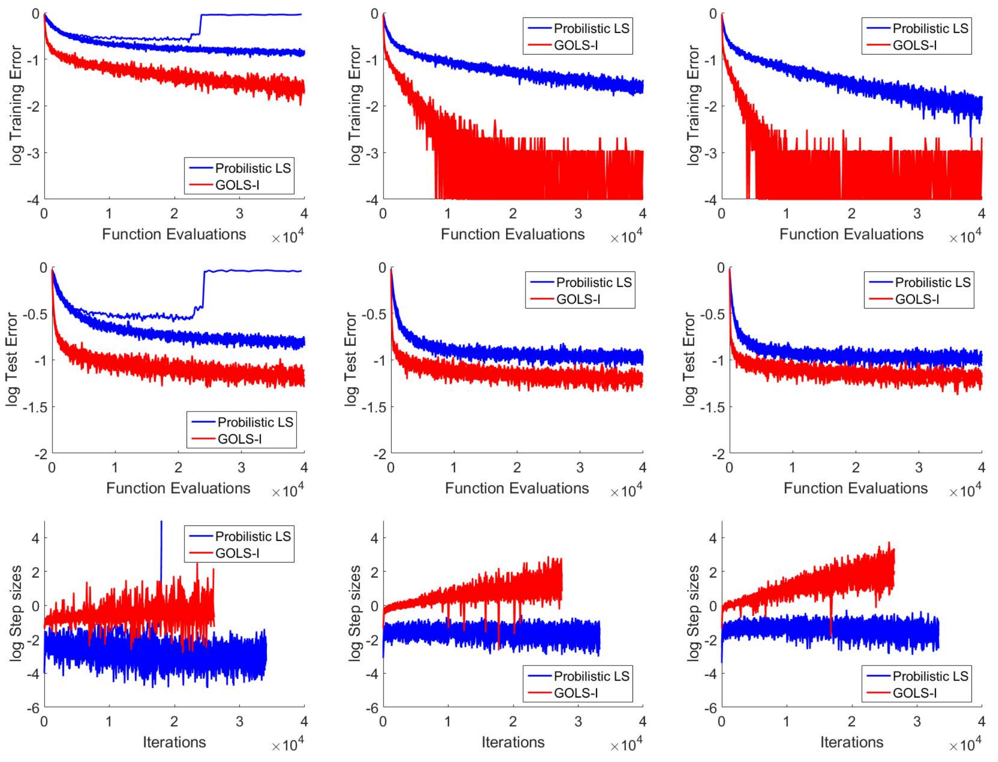
(a) $|\mathcal{B}_{n,i}| = 10$ (b) $|\mathcal{B}_{n,i}| = 50$ (c) $|\mathcal{B}_{n,i}| = 100$ (d) $M = 400$

Figure 4.9: Log training error, log training loss, log test error and the log of the step sizes as obtained with various batch sizes for the logistic regression problem for the BCWD dataset.

The log of the training loss for this problem gives a better perspective of the convergence behaviour of GOLS-I compared to constant step sizes. The training classification errors, as shown in Figure 4.8, hides the absolute performance of the training algorithm, due to rounding the network output to obtain the classification. By considering the training loss, we can observe the training performance in more detail, ignoring the effect of rounding. For $|\mathcal{B}_{n,i}| = 10$, the variance in the computed gradient between batches is high, hindering the performance of GOLS-I in comparison to the small and medium fixed step sizes. As the batch size increases to $|\mathcal{B}_{n,i}| \geq 50$, the quality of the computed gradient improves sufficiently, such that GOLS-I fits the model to the training data up to numerical accuracy within 100, 000 function evaluations. This point occurs earlier in training, as the mini-batch size is increased.

The step size range of 15 orders of magnitude available to GOLS-I is immediately evident in Figure 4.9. At the beginning of training, the step sizes are tightly bound. However, as training progresses, the variance in step sizes increases for $|\mathcal{B}_{n,i}| \in \{10, 50, 100\}$. This occurs as the decision boundary stabilizes in the classification problem. Samples close to the decision boundary exhibit large gradients and prompt small update steps, others are far from the decision boundary and have small gradients, which prompt GOLS-I to compensate and take large step sizes to find a sign change. Depending on the samples in the mini-batch used to construct the search direction, it is also possible to encounter ascent directions. In such cases, GOLS-I decreases the step size until α_{min} is reached. However, in the smooth and continuous case of $M = 400$, the only inaccuracies introduced, are due to GOLS-I's inexact step size resolution, causing the step size rise to be rapid and comparatively consistent between runs. The growing step size behaviour is an adjustment of the line search to the magnitude of the gradient decreasing as the algorithm approaches an optimum. From $10,000$ function evaluations onwards, the variance in step size is due to computational inaccuracy as the gradient approaches numerical 0. At the same point, the loss ceases to decrease during training. This example demonstrates that GOLS-I outperforms constant fixed step sizes and that it generalizes naturally from highly discontinuous, stochastic loss functions to smooth loss functions, with performance increasing as the mini-batch size increases.

4.6.2 The MNIST Dataset with NetI and NetII

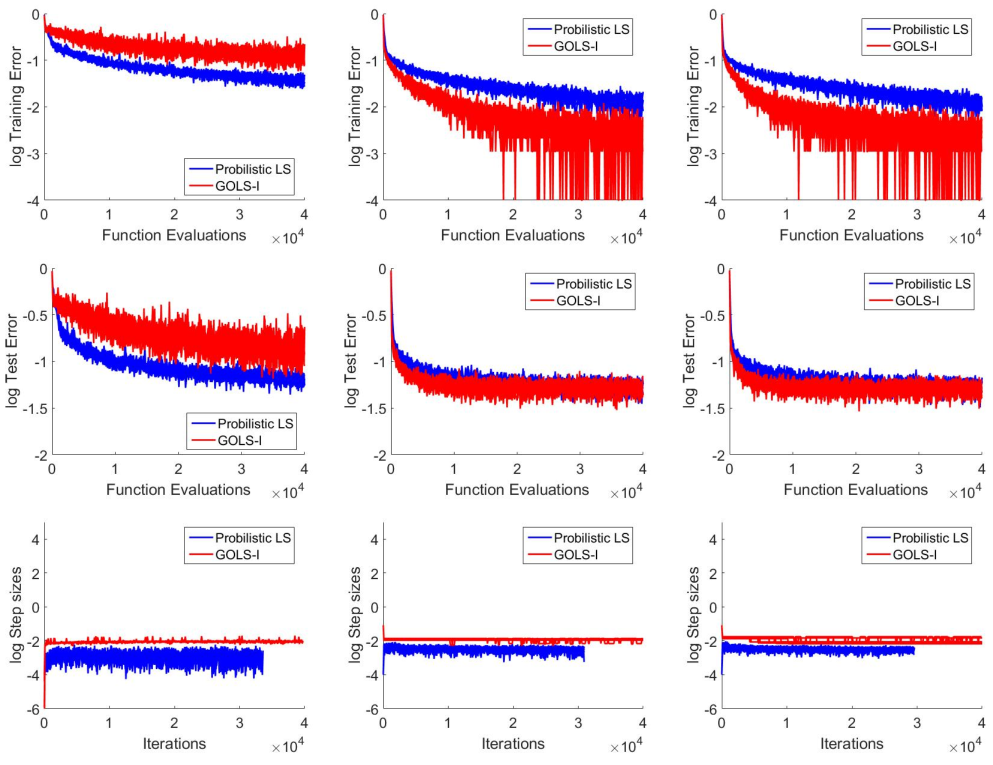


(a) $|\mathcal{B}_{n,i}| = 10$ (b) $|\mathcal{B}_{n,i}| = 100$ (c) $|\mathcal{B}_{n,i}| = 200$

Figure 4.10: log Training error, log training loss, log test error and the log of the step sizes as obtained with various batch sizes for the MNIST dataset with the NetI architecture.

Subsequently, we compare GOLS-I and PrLS as applied to the MNIST dataset with the NetI architecture. The results are given in Figure 4.10 for $|\mathcal{B}_{n,i}| \in \{10, 100, 200\}$. For this example, GOLS-I convincingly outperforms PrLS in both training and test error across all considered mini-batch sizes. PrLS again exhibits divergent behaviour for $|\mathcal{B}_{n,i}| = 10$, while GOLS-I remains stable. There is a significant increase in training performance with GOLS-I, as the mini-batch size increases to $|\mathcal{B}_{n,i}| = 100$. Around 800 function evaluations, some of the first runs obtain zero training classification errors. For $|\mathcal{B}_{n,i}| = 200$, this point is reached around 500 function evaluations. Though training with PrLS also improves as the mini-batch size increases, the average training error reaches a minimum of the order $1 \cdot 10^{-2}$.

As was the case with the BCWD dataset, GOLS-I computes larger step sizes than PrLS while remaining stable during training, which shows that GOLS-I's step sizes are representative of the problem, while PrLS remains conservative. For MNIST on NetI, the number of iterations performed by GOLS-I is visibly less than those of PrLS, indicating that PrLS more readily accepts the initial guess, while not refining the step size to determine a larger step size along the search direction. There is also a difference in step size behaviour between PrLS and GOLS-I: The step sizes determined by PrLS stagnate and even tend to decrease as training progresses, while GOLS-I's step sizes increase during training, which is a function of decreasing gradient magnitudes closer to an optimum. Another noteworthy observation is that both GOLS-I and PrLS automatically determine larger step sizes at the first iteration, as the mini-batch size increases. This is particularly evident between the beginning step sizes determined for $|\mathcal{B}_{n,i}| = 10$ and $|\mathcal{B}_{n,i}| = 100$ respectively, which is consistent with the constant step size analyses in Figure 4.9, where larger step sizes can be more effective with larger mini-batches. The observations of Figure 4.10(last row) indicate that both GOLS-I and PrLS are able to identify this relationship, albeit that PrLS remains more conservative.



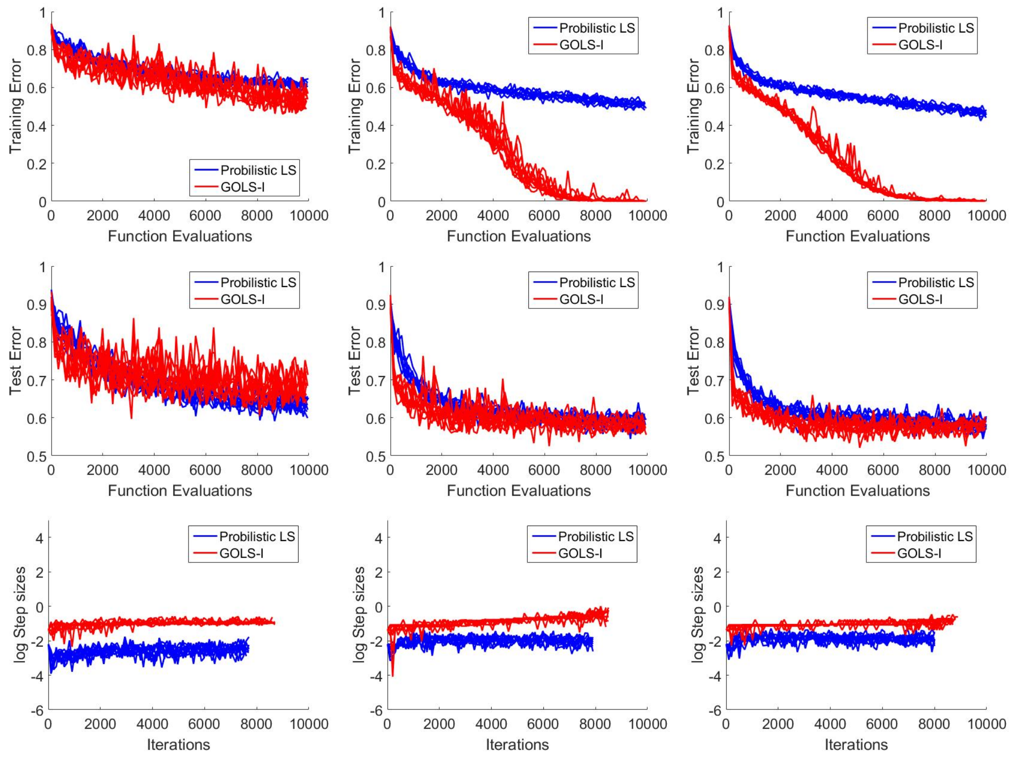
(a) $|\mathcal{B}_{n,i}| = 10$ (b) $|\mathcal{B}_{n,i}| = 100$ (c) $|\mathcal{B}_{n,i}| = 200$

Figure 4.11: Log training error, log training loss, log test error and the log of the step sizes as obtained with various batch sizes for the MNIST dataset with the NetII architecture.

However, this conservatism can have potential benefits, as is demonstrated in Figure 4.11 by the training runs conducted on MNIST with NetII for $|\mathcal{B}_{n,i}| = 10$. Here PrLS has superior performance to GOLS-I, since GOLS-I often accepts the initial step size, evidenced by the number of iterations performed almost matching the number of function evaluations. Since this trend is the same over the different batch sizes, this indicates that the gradient norms of the problem are reasonably consistent over $|\mathcal{B}_{n,i}|$. In general, GOLS-I again determines larger step sizes than PrLS. However, since the quality of the search direction drops with decreasing $|\mathcal{B}_{n,i}|$, the larger step sizes determined by GOLS-I for $|\mathcal{B}_{n,i}| = 10$ lead to noisy, slow training in comparison to PrLS. The smaller step sizes determined by PrLS for $|\mathcal{B}_{n,i}| = 10$ results in more stable, consistent training. It is likely, that the additional information used by PrLS works to its advantage over GOLS-I, when the quality of information is poor.

However, as the batch size increases to $|\mathcal{B}_{n,i}| \in \{100, 200\}$ the search direction and directional derivative quality improve sufficiently, such that GOLS-I again convincingly outperforms PrLS in training. In turn, the test classification errors are comparable between the two line search methods, indicating that both aid in constructing trained models, albeit that GOLS-I trains the model more efficiently. It is important to note, that the step sizes determined by both algorithms are smaller than those of NetI, and remain constant throughout training for both line searches. This indicates that both line search methods are adapting the estimated step sizes to the information presented by the loss function. The comparative number of iterations performed by PrLS is lower for NetII than for NetI, indicating that PrLS required more function evaluations to estimate the step sizes for each iteration, instead of accepting the initial guess, which was the dominant mode of operation for GOLS-I in this example.

4.6.3 The CIFAR10 dataset with NetI and NetII



(a) $|\mathcal{B}_{n,i}| = 10$ (b) $|\mathcal{B}_{n,i}| = 100$ (c) $|\mathcal{B}_{n,i}| = 200$

Figure 4.12: Training error, test error and the log of step sizes as obtained with various batch sizes for the CIFAR10 Dataset, as used with the NetI architecture. The training and test errors are shown on a linear scale to allow comparison with results presented by Mahsereci and Hennig [2017].

The training and test classification errors, with corresponding step sizes for CIFAR10 with NetI are shown in Figure 4.12. To be consistent with results presented in Mahsereci and Hennig [2017], the training and test error plots are plotted on a linear scale for all CIFAR10 analyses. For $|\mathcal{B}_{n,i}| = 10$, GOLS-I marginally outperforms PrLS in training. This is again due to GOLS-I determining larger step sizes than PrLS, which also results in noisier training and higher test errors, due to large updates with inconsistent search directions. Again, the conservative step sizes of PrLS aids stability while training with the large discontinuities present for $|\mathcal{B}_{n,i}| = 10$. However, as mini-batch size increases to $|\mathcal{B}_{n,i}| \in \{100, 200\}$, the performance of GOLS-I again improves significantly, while the improvement for PrLS is marginal. For $|\mathcal{B}_{n,i}| = 100$ GOLS-I trains NetI to within 10% training classification error within 5,000 function evaluations, while PrLS manages 50% at best after 10,000 function evaluations. It is to be noted, that the high test classification errors of the CIFAR10 analyses are due to the training dataset containing only Batch1, a 5^{th} of the total available training data for this problem. Therefore, the networks used for this dataset overfit readily, and do not generalize well. In terms of step sizes, PrLS again remains conservative in comparison to GOLS-I. However, the difference between step sizes determined by GOLS-I and PrLS decreases as the mini-batch size increases. Another notable trend, is that the number of iterations performed by PrLS increases as a function of mini-batch size. This indicates, that the initial accept condition of PrLS is more often satisfied, when more information is available. PrLS also exhibits a notable jump of an order of magnitude in step size between $|\mathcal{B}_{n,i}| = 10$ to $|\mathcal{B}_{n,i}| = 100$. Though it is clear, that PrLS adjusted to the increased quality of information afforded by larger mini-batch sizes, it does not lead to the same increase in performance compared to GOLS-I.

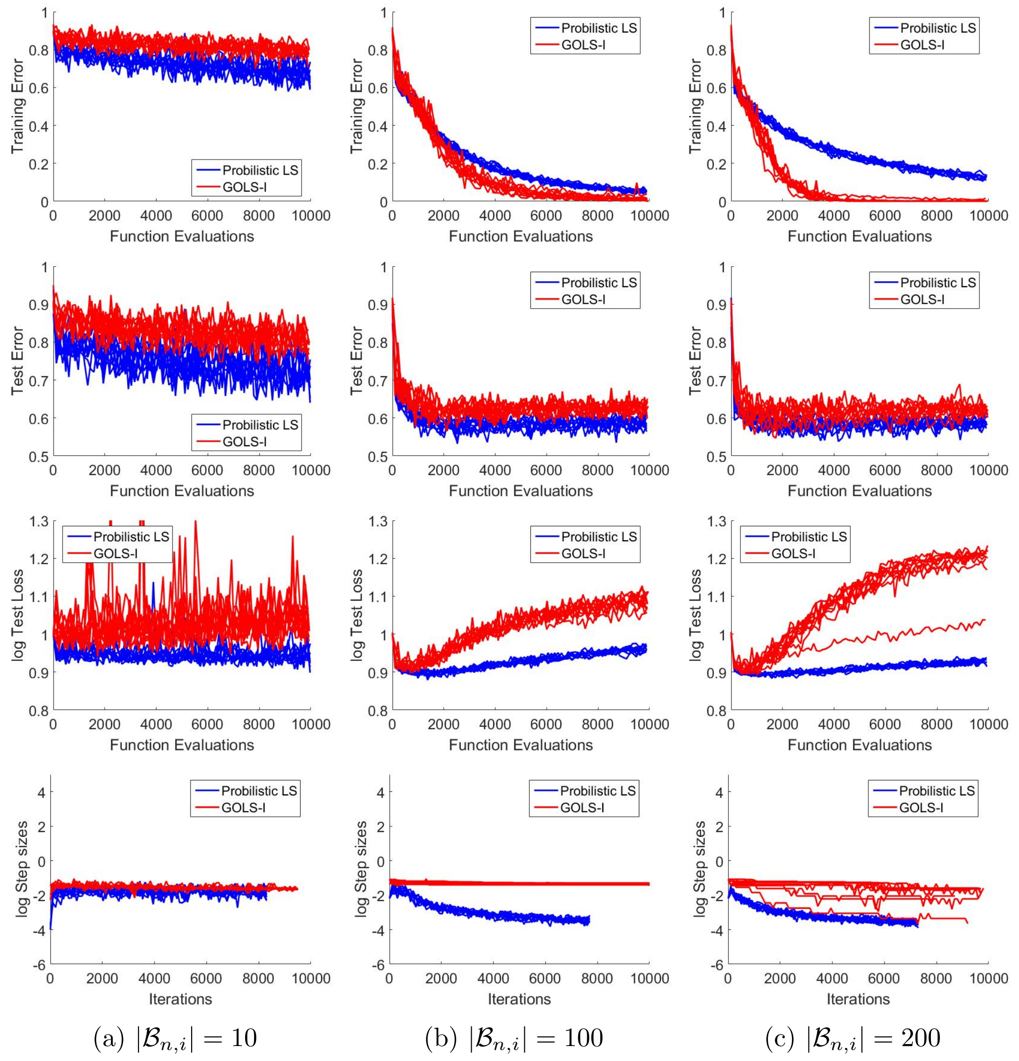


Figure 4.13: Training error, test error, log of test loss and the log of step sizes as obtained with various batch sizes for the CIFAR10 Dataset, as used with the NetII architecture. The training error is shown on a linear scale to allow comparison to results produced by Mahsereci and Hennig [2017].

Consider the results for CIFAR10 with NetII, shown in Figure 4.13. Here, the conservatism of PrLS again benefits to training with $|\mathcal{B}_{n,i}| = 10$. Though the step sizes determined by GOLS-I and PrLS are less than an order of magnitude apart (the closest seen in our investigations), those of PrLS are still on average smaller than those of GOLS-I. This has the implication, that PrLS outperforms GOLS-I in training and test classification errors on NetII with $|\mathcal{B}_{n,i}| = 10$. In turn, there is a significant increase in the performance of GOLS-I with $|\mathcal{B}_{n,i}| \in \{100, 200\}$, while simultaneously PrLS remains competitive with GOLS-I for $|\mathcal{B}_{n,i}| \geq 100$. The test error of PrLS is lower than that of GOLS-I, which is related to: 1) the training data, 2) the network architecture, and 3) the rate at which training occurs. Apart from training only with Batch1, which allows overfitting to occur readily with NetI, NetII has significantly higher flexibility due to having 3 hidden layers. This further increases the ability of NetII to overfit to the incomplete training data. The rate at which overfitting occurs is directly correlated to training performance. To highlight this aspect, we include the test loss in the third row of Figure 4.13. With $|\mathcal{B}_{n,i}| \in \{100, 200\}$, training occurs so rapidly, that the minimum test loss (the point at which overfitting begins) occurs within the first 1,000 function evaluations. As training continues, the training algorithms quickly move past solutions which generalize well, to solutions which minimize the training loss. Consequently, both the test loss and test classification errors continue to increase. Slower training results in the algorithm remaining around solutions which generalize for a larger number of iterations, resulting in lower test loss and test classification errors. This is the case for PrLS for this example, which trains slowly and therefore delays overfitting.

This observation is supported by considering the step sizes shown in the last row of Figure 4.13. For mini-batch sizes $|\mathcal{B}_{n,i}| \in \{100, 200\}$, the step sizes determined by GOLS-I remain constant, while those of PrLS decrease three orders of magnitude from $\alpha_{n,I_n} \approx 1 \cdot 10^{-1}$ to $\alpha_{n,I_n} \approx 1 \cdot 10^{-4}$ during training. This contributes to the slower training behaviour of PrLS, as well as the less aggressive overfit. Unfortunately, this performance can not be attributed to PrLS adapting to the data imbalance of the problem, since the line search is only exposed to the training data, for which it is tasked with minimizing the error. Since PrLS is less successful than GOLS-I in doing so, its lower test error is purely coincidental, given the unique qualities of the problem.

The results of $|\mathcal{B}_{n,i}| = 200$ show an unexpected slowing of training for PrLS compared to $|\mathcal{B}_{n,i}| = 100$. This is confirmed by the test losses, where PrLS overfits faster with $|\mathcal{B}_{n,i}| = 100$ than with $|\mathcal{B}_{n,i}| = 200$. Conversely, the larger mini-batch size once more favours GOLS-I, resulting in efficient training for $|\mathcal{B}_{n,i}| = 200$. This behaviour by PrLS is remarkable, since its step size schedule seems unchanged compared to training with $|\mathcal{B}_{n,i}| = 100$. Therefore, the only other factor is the quality of the search direction. We postulate, that the variance in search direction allowed PrLS to overcome local optima more effectively with $|\mathcal{B}_{n,i}| = 100$, while $|\mathcal{B}_{n,i}| = 200$ causes local optima to be more defined. In combination with its conservative step sizes, this slows the training progress of PrLS. The same occurs to one training run with GOLS-I, which determines much smaller step sizes. However, this occurs only after 1000 iterations, at which point the neural network has already overfit, leading to no obvious deficit in training classification.

In all 5 problems considered, the step sizes were adapted dynamically by both GOLS-I and PrLS. Overall, we find PrLS to be more conservative than GOLS-I in terms of step size magnitude. We find PrLS to be unstable at $|\mathcal{B}_{n,i}| << M$ and full-batch analyses with the BCWD dataset, but superior in training for MNIST and CIFAR10 datasets, when $|\mathcal{B}_{n,i}| = 10$. In the latter case, we postulate that this is due to the use of added information in the form of function value and gradient variance in highly discontinuous loss functions. However, the absolute training performance PrLS with $|\mathcal{B}_{n,i}| = 10$ remains underwhelming in comparison to the performance gain offered by GOLS-I with $|\mathcal{B}_{n,i}| \geq 100$. For $|\mathcal{B}_{n,i}| \in \{100, 200\}$, the conservatism of PrLS leads to slower training performance, while GOLS-I gains a significant performance advantage from the increased quality of gradient information, which causes SNN-GPPs to be more localized. Additionally, GOLS-I does not require the evaluation of variance estimates, nor the construction of surrogates at every iteration.

4.7 Differences to the original PrLS study

Considerable effort went into following the information given in Mahsereci and Hennig [2017] and implementing PrLS as prescribed, especially with regards to constructing the required loss and gradient variance estimates. However, there were some elusive differences between results presented in Mahsereci and Hennig [2017] and our investigations, of which we have not yet found the origin.

Firstly, we were unable to match the resolution obtained in log classification error plots, particularly those of the BCWD dataset problems. This is due to the small number of samples in the dataset. Particularly for the test classification errors, we observe plots with discrete accuracy values, due to the binary nature of individual points being either correctly or falsely classified. Though this is less notable in larger datasets, the same characteristics are present.

Secondly, for the most part, we were not able to reproduce the same training behaviour for PrLS as shown in Mahsereci and Hennig [2017]. Their work shows training and test errors that drop rapidly, then plateau in the log domain for most of their investigated training problems. We do not recover this behaviour for PrLS nor for GOLS-I. Instead, we observed linear convergence in the log domain for both methods, which is consistent with theoretical convergence estimates for SGD [Dekel et al., 2012, Li et al., 2014].

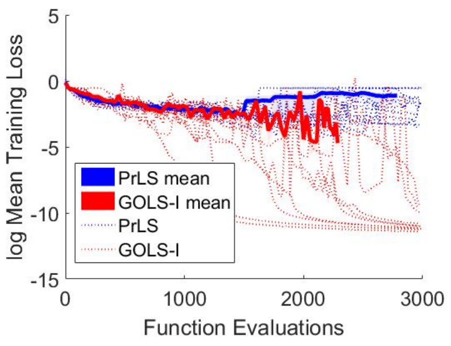
And lastly, though our results for training CIFAR10 with PrLS most closely match those demonstrated in Mahsereci and Hennig [2017], we observe some inconsistencies. For our implementations of NetI, we obtain clearly inferior training performance with PrLS compared to those shown in Mahsereci and Hennig [2017]. Conversely, the training of NetII with PrLS is superior with $|\mathcal{B}_{n,i}| = 100$ and competitive with $|\mathcal{B}_{n,i}| = 200$ to those presented by Mahsereci and Hennig [2017]. By experimentation with our numerical parameters, we noted that the given problems are sensitive to the initial guesses of the neural network weights. However, these are not stated in detail in [Mahsereci and Hennig, 2017]. We therefore suspect, that this might be a potential contributor to some of the encountered discrepancies. We highlight these differences in the interest of reproducible science, not distracting from the PrLS method itself, as it clearly serves its intended purpose in determining step sizes. All parameters were kept constant in this study between PrLS and GOLS-I, with the only difference being the actual line searches used. Although there are differences between our results, and those obtained for PrLS in Mahsereci and Hennig [2017], the comparisons in our investigations hold for the parameters presented in this study.

4.8 PrLS and GOLS-I: Line searches, not update rules

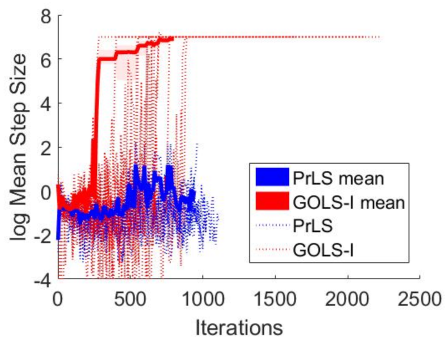
While training MNIST with NetII and CIFAR10 with NetI and NetII, the resulting step sizes for both PrLS and GOLS-I showed that the immediate accept condition was operating more often than the line search method itself. This can be deduced from the number of function evaluations being close to the number of iterations performed. This gives the impression that these line search methods function solely as "update rules", choosing step sizes based on a heuristic, but not actively resolving step sizes. We counter this claim with a demonstration: We select the NetPI architecture with the BCWD dataset and extend the number of hidden layers with 32 nodes from 1 to 10. All other parameters remain the same as those used in Section 4.6.1 for NetPI, where training runs are limited to 3,000 function evaluations. We remind the reader, that the first initial guess for each training run is $\alpha_{0,0} = 10^{-4}$ for PrLS and $\alpha_{0,0} = 10^{-8}$ for GOLS-I respectively. The large number of hidden layers increases the non-linearity of the model, increasing the complexity of the loss function. To ensure that the loss function is discontinuous, while containing sufficient information, the mini-batch size of $|\mathcal{B}_{n,i}| = 100$ is chosen. The resulting training loss, step size, training classification error and test classification error are shown in Figure 4.14. We show individual results for each of the 10 performed training runs in thin, dotted lines, while highlighting the mean performance over the lowest common number of iterations between runs in a thick, solid line.

The higher non-linearity in the loss function makes optimization within the loss landscape more sensitive to the determined step size. Accuracy of the step size may be more critical in such cases, as the penalty in loss for drastically overshooting an optimum may be more severe. Therefore, the onus is on PrLS and GOLS-I to adjust their respective step sizes to the increased non-linearity in the stochastic loss function. Repeated adjustment of step sizes is indicated by an increased number of function evaluations performed per iteration within the line search. The increased non-linearity decreases the probability of the initial guess from the previous iteration being appropriate for the current iteration, reducing the likelihood of the initial accept conditions being satisfied.

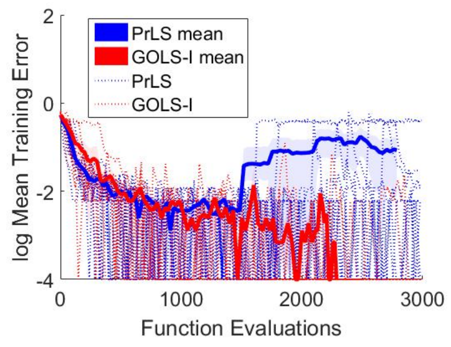
To shed light on the actions taken by the line searches, we summarize the minimum, maximum and mean number of function evaluations (Fe.) performed per iteration (It.) during training for PrLS and GOLS-I in Table 4.3. The minimum number of Fe./It. is indicative of the



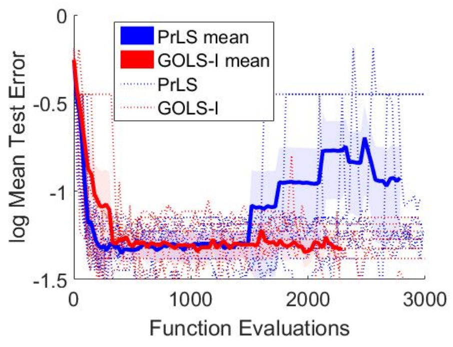
(a) Training loss



(b) Step size



(c) Training classification error



(d) Test classification error

Figure 4.14: Training loss, step size, training classification error and test classification error for the Cancer dataset problem with a modified NetPI architecture. We extend NetPI to contain 10 hidden layers, thereby making the training problem more non-linear, which prompts the line search algorithms to perform more function evaluations per iteration.

immediate accept condition being triggered, while the maximum number of Fe./It. is indicative of the "effort" exerted by the line search to determine the step size for a given iteration. For PrLS, this number is capped to 6(+1) as prescribed by Mahsereci and Hennig [2017]. GOLS-I adjusts the step size repeatedly, until either a sign change in directional derivative is found, or one of the min/max step size limits is reached. This allows for a significantly larger number of Fe./It. to be performed at each iteration. Hence, the maximum number of Fe./It. being performed in the first iteration by GOLS-I is 28, as it adjusts the step size from the conservative initial guess of $\alpha_{0,0} = \alpha_{min} = 1 \cdot 10^{-8}$ to an appropriate magnitude of around $\alpha_{0,I_n} = 1 \cdot 10^{0}$. In subsequent iterations after initialization, the maximum number of Fe./It. performed by GOLS-I during continued training for this problem was 11. This indicates that both PrLS and GOLS-I were capable of continually adjusting step sizes according to the requirements of the loss landscape. This assertion is also supported by the significant variance in step sizes early on in training. This results in a lower number of iterations performed during the training runs for a maximum 3,000 function evaluations. Generally, the number of Fe./It. drops as training progresses, resulting in lower overall Fe./It. averages. This applies in particular for GOLS-I, where the directional derivatives encountered at the step size upper limit are negative, triggering the initial accept condition.

This investigation demonstrates, that both PrLS and GOLS-I adapt to the loss function characteristics presented to them and can be considered functional line searches in discontinuous loss functions. As is consistent with the previous analyses conducted in this chapter, GOLS-I shows improved performance over PrLS for the given example. GOLS-I is able to reduce the loss to $1 \cdot 10^{-10}$ for isolated training runs as well as obtain zero training classification errors, which is not the case with PrLS. Instead, PrLS exhibits divergent training behaviour after 1500 function evaluations on average.

Overall, GOLS-I has been shown to be more aggressive than PrLS in terms of the step

	Min Fe./It.	Max Fe./It.	Mean Fe./It.
GOLS-I	1	28 (11)	1.3
PrLS	1	7	2.7

Table 4.3: Various metrics of function evaluations (Fe.) performed per iteration (It.) during training for PrLS and GOLS-I. Function evaluations performed during an iteration indication of the "effort" exerted by the line search to determine the step size. The maximum number of Fe./It. in PrLS is fixed, which is not the case for GOLS-I. The absolute maximum number of Fe./It. for GOLS-I is 28, which occurs in the first iteration due to a conservative initial guess of $\alpha_{0,0} = 1 \cdot 10^{-8}$. In subsequent training, the maximum number is 11.

size magnitudes, which can lead to detrimental performance when small mini-batches are used. However, given the hardware capabilities currently available to machine learning practitioners, it is feasible to implement mini-batch sizes of $|\mathcal{B}_{n,i}| \geq 100$, where GOLS-I has demonstrated superior performance over PrLS for the problem considered. We also remind the reader, that no variance estimates or surrogates are needed to implement GOLS-I, making its application to existing machine learning technologies less involved and computationally more efficient than PrLS.

4.9 Conclusion

For discontinuous dynamic mini-batch sub-sampled (MBSS) loss functions, we compare the Gradient-Only Line Search that is Inexact (GOLS-I) [Kafka and Wilke, 2019b], to the Probabilistic Line Search (PrLS) [Mahsereci and Hennig, 2017] for automatically resolving learning rates. GOLS-I is an intuitive, computationally efficient alternative line search method, which does not require surrogates or function value and gradient estimates, while remaining robust in discontinuous loss functions. Instead of minimizing or finding critical points along descent directions, GOLS-I locates Stochastic Non-Negative Associated Gradient Projection Points (SNN-GPPs). Moving along a 1-D descent direction, SNN-GPPs are identified by sign changes from negative to positive in the directional derivative, thus incorporating second order information representative of a minimum.

We demonstrate the capabilities of GOLS-I on eight machine learning problems, with five proposed by Mahsereci and Hennig [2017], three adapted from Prechelt [1994]. These include the Breast Cancer Wisconsin Diagnostic (BCWD) dataset with four different architectures, as well as MNIST and CIFAR10 each implemented with a shallow and deep network architecture. We use these problems to demonstrate that step sizes can be efficiently determined for stochastic gradient descent with a line search (LS-SGD) using GOLS-I. GOLS-I adaptively determines step sizes that can vary over 15 orders of magnitude, i.e. from a minimum step size of $\alpha_{min} = 10^{-8}$ to a maximum of $\alpha_{max} = 10^{7}$. In our experiments, GOLS-I demonstrated training performance that is competitive to superior to that of a manually tuned constant step size. As training progressed, GOLS-I was able to dynamically re-adjust the step size, depending on the characteristics of the loss function.

We have shown GOLS-I to outperform the probabilistic line search (PrLS) in training when mini-batches are sufficiently large, while the performance of PrLS was complementary to GOLS-I, in that it performed best with small mini-batch sizes. The combination of PrLS being more conservative than GOLS-I in the step sizes, as well as the added information used by PrLS (in the form of loss function and gradient variance estimates) on average makes PrLS more robust than GOLS-I for the MNIST and CIFAR10 problems with mini-batch sizes $|\mathcal{B}_{n,i}| = 10$. For the BCWD dataset, PrLS exhibited divergent behaviour with $|\mathcal{B}_{n,i}| = 10$ and some instances full-batch training. However, using mini-batch sizes of $|\mathcal{B}_{n,i}| = 10$ results in slow training overall. Current computational resources support mini-batch sizes of sufficient size ($|\mathcal{B}_{n,i}| \geq 100$), such that gradient information alone, in the form of the SNN-GPP as used by GOLS-I, is sufficient to enable effective step sizes to be determine in dynamic MBSS loss functions. In such cases, GOLS-

I comprehensively outperformed PrLS in this investigation, which consists largely of problems and architectures proposed by the authors of PrLS. This makes GOLS-I a credible alternative to determining learning rates in dynamic mini-batch sub-sampled loss functions, which leads to curiosity regarding the feasibility of incorporating GOLS-I into other traditional mathematical programming methods and popular neural network training algorithms.

Chapter 5

Gradient-only line searches to automatically determine learning rates in stochastic training algorithms

Gradient-only and probabilistic line searches have recently reintroduced the ability to adaptively determine learning rates in dynamic mini-batch sub-sampled neural network training. However, stochastic line searches are still in their infancy and thus call for ongoing investigation. We study the application of the *Gradient-Only Line Search that is Inexact* (GOLS-I) to automatically determine the learning rate schedule for a selection of popular neural network training algorithms, including NAG, Adagrad, Adadelta, Adam and LBFGS, with numerous shallow, deep and convolutional neural network architectures trained on different datasets with various loss functions. We find that GOLS-I's learning rate schedules are competitive with manually tuned learning rates, over seven optimization algorithms, three types of neural network architecture, 23 datasets and two loss functions. We demonstrate that algorithms, which include dominant momentum characteristics, are not well suited to be used with GOLS-I. However, we find GOLS-I to be effective in automatically determining learning rate schedules over 15 orders of magnitude, for most popular neural network training algorithms, effectively removing the need to tune the sensitive hyperparameters of learning rate schedules in neural network training.

5.1 Introduction

In pursuit of improving the training of neural networks, learning rate parameters are consistently some of the most sensitive hyperparameters in deep learning [Bergstra and Bengio, 2012], and hence constitute an ongoing area of research [Smith, 2015, Orabona and Tommasi, 2017, Wu et al., 2018]. The field of mathematical programming predominantly employs minimization line searches to determine step sizes (or learning rates) [Arora, 2011]. However, ever since neural network training has moved from full-batch to mini-batch sub-sampled (MBSS) training in order to improve computational cost and training characteristics, line searches have fallen out of favour [Wilson and Martinez, 2003, Schraudolph and Graepel, 2003, Schraudolph et al., 2007]. The prohibiting factor in successfully implementing minimization line searches has been the emergence of discontinuities in the loss functions and gradients, that are due to continuous resampling of mini-batches, see Figure 5.1. Subsequently, the primary research focus in stochastic minimization has been on various *a priori* selected step sizes [Schraudolph, 1999, Boyd et al., 2003, Smith, 2015].

However, recently, line searches were reintroduced to dynamic MBSS loss functions in neural network training in the form of the probabilistic line search by Mahsereci and Hennig [2017]. This method deals with the discontinuities in function value and gradients by constructing Gaus-

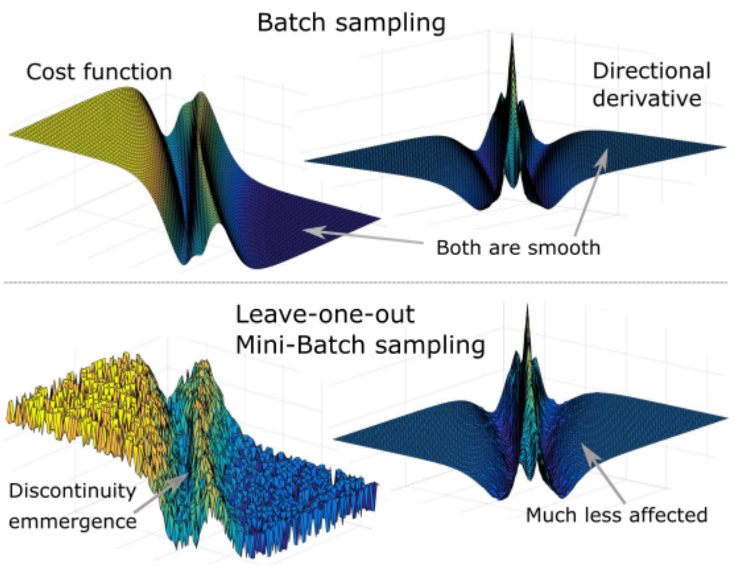


Figure 5.1: Dynamic mini-batch sub-sampling introduces discontinuities into the loss and gradients of a neural network. However, gradient information, depicted by the directional derivative along a unit direction of two weights, is less affected by discontinuities during resampling. Gradient-only line searches exploit this property to determine step sizes with the aid of the gradient-only optimality condition [Wilke et al., 2013, Snyman and Wilke, 2018, Kafka and Wilke, 2019b, Kafka and Wilke, 2019].

sian process surrogates along search directions. The minimum of the surrogate can subsequently be accurately determined. Alternatively, *gradient-only* line search methods [Kafka and Wilke, 2019b] employ concepts based on gradient-only optimization [Wilke et al., 2013, Snyman and Wilke, 2018] to determine step sizes. This is achieved by locating Stochastic Non-Negative Associated Gradient Projection Points (SNN-GPPs), which can be distinct from function minimizers in discontinuous functions. An SNN-GPP manifests as a sign change in directional derivative from negative to positive along a search direction. This means, that the SNN-GPP definition includes second order information, allowing the line search to avoid maxima and inflection points, searching only for minima that are present in the derivative, thereby reducing the number of candidate solutions. The use of the gradient-only paradigm has a double benefit: 1) Gradient information is less affected by discontinuities present in dynamic MBSS loss functions (see Figure 5.1) and 2) the spatial location of SNN-GPPs remains bounded. When implemented, these factors have resulted in a gradient-only line search formulation, which has been shown to outperform probabilistic line searches in their current form [Kafka and Wilke, 2019].

In this chapter we investigate the suitability of using the *Gradient-Only Line Search that is Inexact* (GOLS-I), see Appendix A.3.2, to determine the learning rate parameters for a collection of seven popular neural network training algorithms on three types of neural network architectures, two loss functions and 23 datasets. This work fits into the larger context of improving optimization performance and efficiency in neural network training.

5.2 Connections: Approaches in advancing neural network training

Neural network training requires an optimization problem to be solved. Two fundamental components to any optimization problem are the formulation of the problem, with the resulting optimization landscape (called the loss function); and the behaviour of the optimizer selected to traverse the given landscape in the search of an optimum (also referred to as the training algorithm). We briefly review both of these aspects in the context of neural network training.

5.2.1 Modifying loss function landscapes

Significant research has been directed towards theoretical characterization of the loss functions in neural network architectures. However, admittedly there is still a gap between theoretical basis and practical experiences in training. Hence there has been movement to recast problems in manners that lessen the void between practice and more rigorously understood optimization theory [Kawaguchi, 2016].

Due to the non-convex nature of neural network loss functions, see Figure 5.1, many optimizers have difficulty finding optima. Therefore, there have been significant attempts to improve the characteristics of the loss function by,

- Loss function scaling for a given model [Ioffe, 2017, Salimans and Kingma, 2016],
- Regularization of a network model for convexity [Bishop, 2006], and
- Modification of a network training problem via hyperparameter optimization [Bergstra et al., 2011].

If length scales of the loss function in different dimensions are similar, the problem may become less sensitive to the training algorithm and learning rate selected. Methods such as both Batch Norm [Ioffe, 2017] and Weight Norm [Salimans and Kingma, 2016] can be used to this effect. Batch Norm specifically is claimed to be effective in correcting the scaling of deep networks, where the exploding or diminishing gradient problems become prevalent in deeper layers, causing vastly differing curvatures along different directions.

A common method of introducing convexity into the loss function landscape is by adding a regularization term such as a quadratic L2-normalization term to the network loss [Bishop, 2006]. Alternatively, work done by Li et al. [2017] shows how architectural decisions, specifically the introduction of skip connections, in a neural network affect the convexity of the loss function. Interesting investigative work done by Goodfellow et al. [2015] shows that networks, which perform well when used with stochastic gradient descent (SGD) often result in convex training paths. However, the authors do show that encountering narrow, flat ravines or flat planes is a possibility. Given this highly non-linear nature of loss functions, it is unlikely that a single fixed learning rate would perform well over all iterations. Goodfellow et al. [2015] also demonstrated that another contributing factor to the performance of the algorithm were the directions generated for a given update step. If the variance in the generated directions is high, SGD follows trajectories which make little progress towards a minimum. This can be especially pronounced when the optimizer is traversing a flat plane with a small learning rate. These considerations are consistent with the observations by Smith et al. [2017], which suggests finding better minima by ramping up the mini-batch size, thereby reducing gradient variance, as opposed to decreasing the learning rate.

Global hyperparameter optimization strategies consider the various parameters involved in problem formulation (network architecture, activation function, loss function etc.) and optimizer (update rule, learning rates etc.). Depending on the problem, there might be combinations of architecture and algorithm parameters that are more effective than others. Hyperparameter optimization approaches seek to isolate such combinations by changing both the loss landscape and optimization parameters simultaneously. Such methods include grid-searches, random search and Bayesian optimization [Bergstra et al., 2011, Bergstra and Bengio, 2012, Snoek et al., 2012]. However, they are generally expensive and require a large number of trial runs to evaluate the fitness of different parameter configurations. Although these methods are used to configure such a wide range of parameters, work done by Bergstra et al. [2011], Bergstra and Bengio [2012] shows that the learning rate parameters are consistently the most sensitive across different problems. It is therefore of interest to determine the learning rate parameters automatically, making the remaining hyperparameter space smaller and easier to optimize.

5.2.2 Improving optimizers

In the interest of constructing an effective optimizers for traversing the complex landscapes of neural network loss functions, research has been focussed on

- a priori selected step sizes [Darken and Moody, 1990], and
- adaptively computed step sizes [Darken and Moody, 1990].

An alternative to line searches is the use of *a priori* determined fixed learning rates and learning rate schedules, forming part of sub-gradient methods [Boyd et al., 2003]. However, early work showed that fixed learning rates (step sizes) that are too small result in inefficient use of computational resources, while parameters that are too large inhibit optimization algorithms from converging to solutions with sufficient accuracy [Moreira and Fiesler, 1995]. Hence, a more popular approach in the deep learning community is to use learning rate schedules [Darken and Moody, 1990, Moreira and Fiesler, 1995, Senior et al., 2013, Vaswani et al., 2017, Denkowski and Neubig, 2017], which change the learning rate according to a predetermined rule. Some schedules focus on incorporating oscillatory behaviour such as learning rate cycling [Smith, 2015], which ramps the learning rate up and down within a given range. A similar approach is the use of warm restarts [Loshchilov and Hutter, 2016]. Both these methods periodically vary their learning rates with regards to a specified function and schedule. This is inspired by allowing the optimization algorithm to balance exploration (high learning rate) and exploitation (small learning rate) to escape local minima or saddle points, and increase the odds of finding good minima. However, in general the primary disadvantage of learning rate schedules is that both the functional form of the schedule, as well as its own hyper-parameters, are problem dependent, but determined *a priori* to training, making generalized application of learning rate schedules difficult. That being said, popular optimizer improvements have included incorporating learning schedules directly into steepest descent algorithms [Duchi et al., 2011, Zeiler, 2012, Kingma and Ba, 2015, Zheng and Kwok, 2017, Wu et al., 2018]. However, in most cases, a learning rate magnitude parameter remains.

Another area of learning rate research are adaptive step size methods, which change parameters based on loss function information presented to the algorithm during training. This category includes line searches, which have been implemented in training machine learning problems in the field of adaptive sub-sampling. Here the emphasis lies on governing qualities of the training algorithm's descent direction through determining the mini-batch size. Once selected, mini-batches remain unchanged over the course of an iteration [Friedlander and Schmidt, 2011, Bollapragada et al., 2017]. However, Bottou [2010] argues, that continually changing mini-batches can benefit training, by exposing the algorithm to increased amounts of information.

Consequently, alternative adaptive step size strategies have been developed, specifically to operate on discontinuous loss functions. Examples of such methods include the incorporation of statistical concepts into existing methods, such as the adaptation of coin betting theory to determine the learning rate [Orabona and Tommasi, 2017] and a statistically motivated adaptation of an annealed learning rate [Pouyanfar and Chen, 2017]. Interest also returned to the well established gradient descent with momentum algorithm [Zhang and Mitliagkas, 2017]. The work by Zhang and Mitliagkas [2017] demonstrated that by adaptively fine tuning the momentum term, performance of gradient descent with momentum can be competitive with state of the art algorithms, while obtaining better generalization. A particularly interesting albeit involved approach has been to use deep learning methods themselves to determine the learning rate during training [Xu et al., 2017, Bello et al., 2017, Andrychowicz et al., 2016]. One approach has been to use a controller recurrent neural network to learn the appropriate update rule for the main network by assembling ingredients from a database of existing update rules [Bello et al., 2017]. Another approach employs reinforcement learning by using the combination of an actor and critic network to learn and determine the learning rate schedule for the main network. This particular method trains 3 networks simultaneously [Xu et al., 2017]. Such approaches can become computationally expensive, while trying to fill the void left by the absence of mainstream adoption of line searches.

On an exploratory note, it would be interesting to consider the effectiveness of Lipschitz global optimization in the context of neural network training [Strongin et al., 2000]. This field has been predominantly applied in engineering applications, where models have high dimensionality and are expensive to evaluate [Kvasov et al., 2012]. Recent interest has even extended these methods to stochastic functions [Aribi et al., 2019], which might be of interest to neural network training. It is notable, that these methods require the Lipschitz constant to be determined, which can be achieved using a variety of means [Strongin et al., 2000, Gaviano and Lera, 2008]. However, in this regard, the SNN-GPP as used in GOLS, has the advantage that it does not strongly depend on the magnitude of the gradient, only its sign along a search direction.

5.3 Our contribution

Typical learning rate schedules generate values that exponentially decay from 0.1 to 10^{-6} [Senior et al., 2013], while the magnitudes of the aforementioned cyclical learning rate schedules vary over 3 or 4 orders of magnitude [Smith, 2015, Loshchilov and Hutter, 2016]. Such schedules can have up to 6 different parameters which need to be set. The *Gradient-Only Line Search that is Inexact* (GOLS-I) [Kafka and Wilke, 2019b] is able to determine step sizes within a maximum range of 15 orders of magnitude and has no tunable hyperparameters. This allows the line search to traverse various loss function features, such as flat planes as well as steep slopes.

We recast various popular training algorithms, namely (Stochastic) Gradient Descent (SGD) [Robbins and Monro, 1951], SGD with momentum (SGDM) [Rumelhart et al., 1988], Nesterov's Accelerated Gradient (NAG) [Nesterov, 1983], Adagrad [Duchi et al., 2011], Adadelta [Zeiler, 2012] and Adam [Kingma and Ba, 2015] to be compatible with line searches. Subsequently, we employ GOLS-I to determine the learning rates for these algorithms to explore whether the line search can adapt step sizes according to the characteristics of a given algorithm. Every algorithm contributes a search direction, for which GOLS-I is then tasked with determining a step size. This presents two questions: 1) Is GOLS-I able to adapt step sizes to different algorithms during training in the context of mini-batch sub-sampled loss functions? And 2) Which directions perform best with a line search, see Figure 5.2. Arguably, none of the selected algorithms (apart from perhaps SGD as a derivative of gradient descent) were designed to be used with a line search, rendering the answer to the latter question non-self-evident.

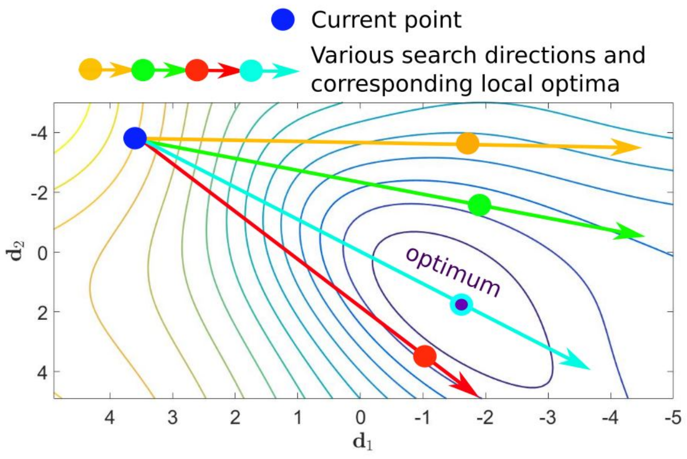


Figure 5.2: Not all directions are created equal. Different assumptions and paradigms result in a plethora of optimization strategies with varying search direction trajectories. Step sizes along these directions can be determined using line searches.

In this chapter we first explore the interaction between GOLS-I and the various algorithms by training 22 different foundational classification problems with both shallow and deep network architectures. Subsequently we implement GOLS-I in PyTorch and apply it to the CIFAR10 dataset with the ResNet18 architecture to demonstrate how GOLS-I interacts with different algorithms on a larger problem.

5.4 A summary of gradient-only line search concepts

Deep learning loss functions are predominantly formulated as

$$\mathcal{L}(\boldsymbol{x}) = \frac{1}{M}\sum_{b=1}^{M} \ell(\boldsymbol{x};\ \boldsymbol{t}_b), \tag{5.1}$$

where $\boldsymbol{x} \in \mathcal{R}^p$ is a vector of network weights, $\{\boldsymbol{t}_1, \ldots, \boldsymbol{t}_M\}$ is a training dataset with M samples, and $\ell(\boldsymbol{x};\ \boldsymbol{t})$ is the loss formulation used to evaluate the performance of weights $\boldsymbol{x}$ with regards to training dataset sample $\boldsymbol{t}$. Backpropagation [Werbos, 1994] is then used to compute the exact gradient w.r.t. $\boldsymbol{x}$ such that:

$$\nabla\mathcal{L}(\boldsymbol{x}) = \frac{1}{M}\sum_{b=1}^{M} \nabla\ell(\boldsymbol{x};\ \boldsymbol{t}_b). \tag{5.2}$$

In the case where the full batch of training data is used to evaluate $\mathcal{L}(\boldsymbol{x})$ and $\nabla\mathcal{L}(\boldsymbol{x})$, the resulting smoothness and continuity is only dictated by the smoothness and continuity properties of the activation functions, see Figures 5.1 and 5.3 where we construct loss functions using the classic Iris [Fisher, 1936] dataset with a simple feed forward neural network with 10 hidden nodes.

A line search can then be conducted along a search direction $\boldsymbol{d}_n$ within the loss function, by constructing one-dimensional function $\mathcal{F}_n$ as a function of step size α as follows:

$$\mathcal{F}_n(\alpha) = f(\boldsymbol{x}_n(\alpha)) = \mathcal{L}(\boldsymbol{x}_n + \alpha\boldsymbol{d}_n), \tag{5.3}$$

with corresponding directional derivative

$$\mathcal{F}'_n = \frac{dF_n(\alpha)}{d\alpha} = \boldsymbol{d}_n^T \cdot \nabla\mathcal{L}(\boldsymbol{x}_n + \alpha\boldsymbol{d}_n), \tag{5.4}$$

where n denotes the iteration of a particular algorithm, currently located at $\boldsymbol{x}_n$. F_n is a univariate search of the multi-dimensional loss function along the search direction $\boldsymbol{d}_n$, where $\mathcal{F}_n$ denotes a specific form of univariate search, where the loss function evaluated is $\mathcal{L}(\boldsymbol{x})$. An example is shown on the top of the middle column of Figure 5.3. A line search then seeks to find the optimum along the search direction of the loss function, see Figure 5.2, which may require a number of function evaluations. We therefore denote step size as α_{n,I_n}, where n remains the iteration number, and I_n is the number of function evaluations required to determine the step size for iteration n. This also generalizes to fixed step sizes (learning rates), where α_{n,I_n} is selected *a priori* and $I_n = 1$.

Minimization line searches have been shown to be effective in smooth, continuous functions [Arora, 2011]. However, modern datasets coupled with large neural network architectures [Krizhevsky et al., 2012] exceed the available memory resources of computational nodes or graphical processing units (GPUs). Consequently, mini-batch sub-sampling (MBSS) is implemented, where only a fraction of the available training data $\mathcal{B} \subset \{1, \ldots, M\}$ of size $|\mathcal{B}| \ll M$ is used to construct an approximate loss:

$$L(\boldsymbol{x}) = \frac{1}{|\mathcal{B}|}\sum_{b\in\mathcal{B}} \ell(\boldsymbol{x};\ \boldsymbol{t}_b), \tag{5.5}$$

and corresponding approximate gradient

$$\boldsymbol{g}(\boldsymbol{x}) = \frac{1}{|\mathcal{B}|}\sum_{b\in\mathcal{B}} \nabla\ell(\boldsymbol{x};\ \boldsymbol{t}_b). \tag{5.6}$$

These approximations then replace $\mathcal{L}(\boldsymbol{x})$ and $\nabla\mathcal{L}(\boldsymbol{x})$ for line searches in Equations (5.3) and (5.4) respectively. Both approximations have respective expectations $\mathbb{E}[L(\boldsymbol{x})] = \mathcal{L}(\boldsymbol{x})$ and $\mathbb{E}[\boldsymbol{g}(\boldsymbol{x})] = \nabla\mathcal{L}(\boldsymbol{x})$ [Tong and Liu, 2005]. However, individual evaluations of $L(\boldsymbol{x})$ and $\boldsymbol{g}(\boldsymbol{x})$ may vary significantly form the expectation, resulting in a sampling bias for each individual mini-batch, and variance between subsequent mini-batches.

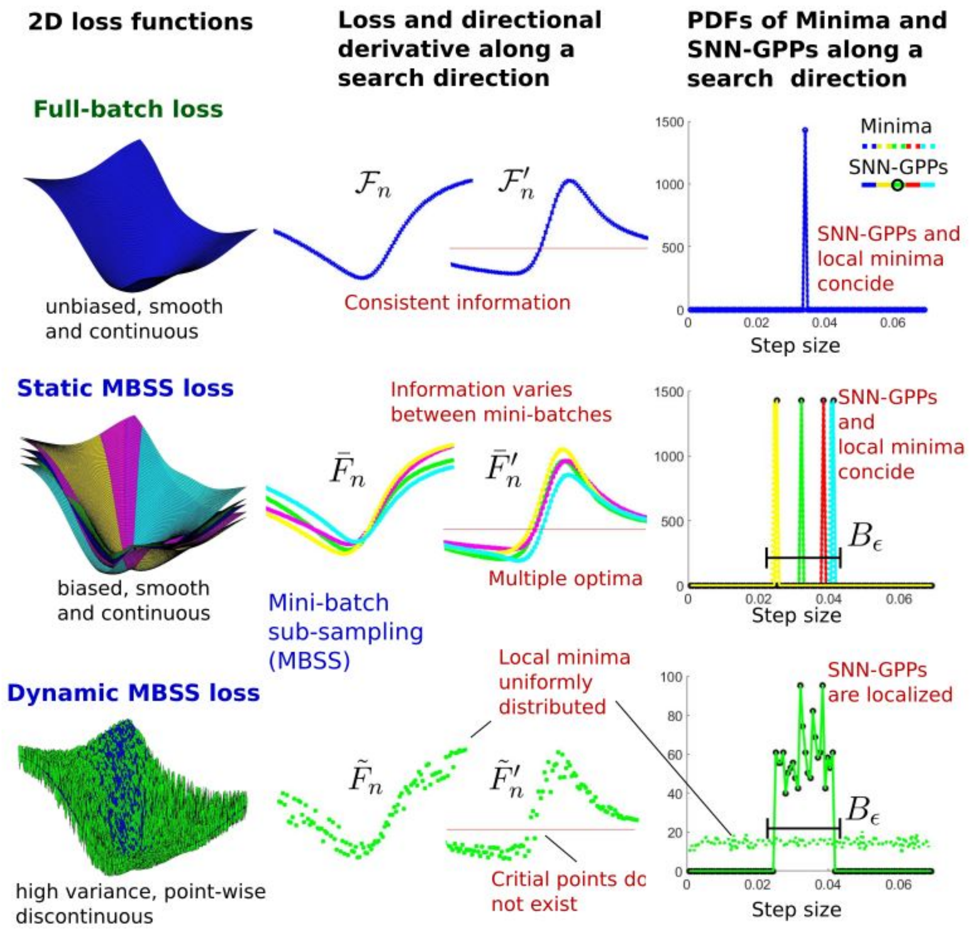


Figure 5.3: Summary of loss surfaces, static and dynamic mini-batch sub-sampling (MBSS) and the performance of minima vs SNN-GPPs.

In the field of adaptive sub-sampling methods, the emphasis lies on determining $|\mathcal{B}|$ or sample constituents of $\mathcal{B}$ that result in desired characteristics of $L(\boldsymbol{x})$. The aim might be to select $\mathcal{B}$, such that $L(\boldsymbol{x}) \approx \mathcal{L}(\boldsymbol{x})$, or to construct $L(\boldsymbol{x})$ such that it on average presents descent directions to the given algorithm [Friedlander and Schmidt, 2011, Bollapragada et al., 2017]. Since a lot of computational effort goes into selecting $\mathcal{B}$ for estimation $L(\boldsymbol{x})$, it is maintained over a minimum duration of a line search at iteration, n, in a given algorithm, before $\mathcal{B}$ is resampled [Byrd et al., 2011, 2012, Martens, 2010]. This approach is referred to as static MBSS, denoted as $\bar{L}(\boldsymbol{x})$, which presents smooth, but biased estimates to the optimization algorithm. A mini-batch sampled using static MBSS is denoted as $\mathcal{B}_n$ [Kafka and Wilke, 2019b]. We give a specific example in Figure 5.3, where we divide the training data into four mini-batches, the content of which remain constant, resulting in four different expressions of $\bar{L}(\boldsymbol{x})$ and $\bar{\boldsymbol{g}}(\boldsymbol{x})$. Corresponding line searches would be denoted $\bar{F}_n$ with derivative $\bar{F}'_n$. This approach allows minimization line searches to be used within given optimization algorithms. However, in our example a line search will find one of four different minimizers, depending on which mini-batch is selected.

An alternative sampling strategy is dynamic MBSS, in which a new sub-sample of training data is selected at every evaluation, i, of $L(\boldsymbol{x})$, subsequently denoted as $\tilde{L}(\boldsymbol{x})$ and $\tilde{\boldsymbol{g}}(\boldsymbol{x})$ with line searches $\tilde{F}_n$ and $\tilde{F}'_n$. Since mini-batches are resampled at every function evaluation, i, they are correspondingly denoted as $\mathcal{B}_{n,i}$. In the example shown in Figure 5.3 (last row), we uniformly sample one of the four mini-batches (previously constructed for the static MBSS example) for each function evaluation. The resulting loss function estimates are discontinuous due to continually changing mini-batches, which breaks the continuity of information between function evaluations. These discontinuities make encountering a critical point, $\tilde{F}'_n(\alpha^*) = 0$ [Arora, 2011], unlikely, even when $\mathcal{F}'_n(\alpha^*) = 0$ for the full-batch loss. Additionally, minimization line searches often wrongly identify discontinuities as local minima [Wilson and Martinez, 2003, Schraudolph and Graepel, 2003, Schraudolph et al., 2007], motivating the aforementioned shift to subgradient methods [Schraudolph, 1999, Boyd et al., 2003, Smith, 2015]. Implementing dynamic MBSS has also been referred to as approximate optimization [Bottou, 2010], having the benefit of exposing algorithms to larger amounts of information, due to continuous re-sampling of data. Line searches were first introduced into dynamic MBSS loss functions by Mahsereci and Hennig [2017] and are subsequently receiving increased attention [Mahsereci and Hennig, 2017, Wills and Schön, 2018, 2019, Kafka and Wilke, 2019a, Kafka and Wilke, 2019].

The GOLS-I method introduced by Kafka and Wilke [2019b] employs the stochastic *gradient-only* equivalent to the optimality criterion in the form of the *Stochastic Non-Negative Associative Gradient Projection Point* (SNN-GPP) [Kafka and Wilke, 2019b], which forms part of gradient-only optimization [Wilke et al., 2013, Snyman and Wilke, 2018]. The work done by Kafka and Wilke [2019b] presents proofs that GOLS-I converges within a bounded ball, B_ϵ, where the ball bounds all possible SNN-GPPs of the surrounding neighbourhood. The SNN-GPP itself is defined as follows:

Definition 5.4.1. *SNN-GPP: A stochastic non-negative associated gradient projection point (SNN-GPP) is defined as any point, $\boldsymbol{x}_{snngpp}$, for which there exists $r_u > 0$ such that*

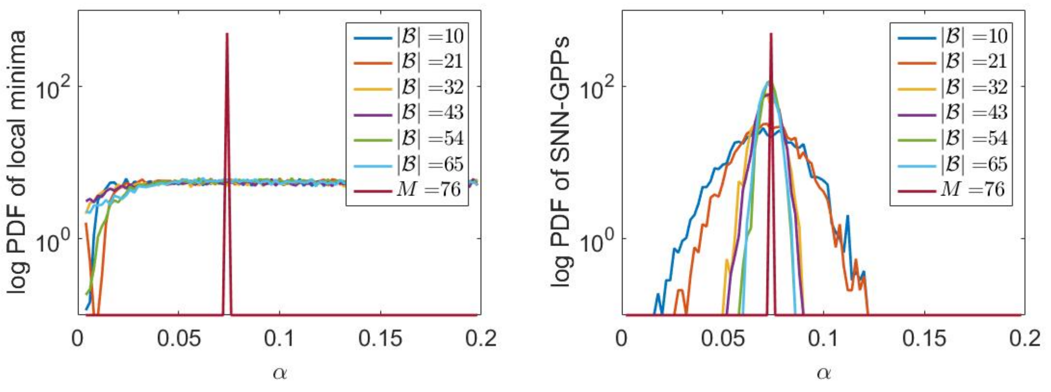
$$\nabla f(\boldsymbol{x}_{snngpp} + \lambda \boldsymbol{u})\boldsymbol{u} \geq 0, \ \ \forall\, \boldsymbol{u} \in \{\boldsymbol{y} \in \mathbb{R}^p \mid \|\boldsymbol{y}\|_2 = 1\}\,, \ \ \forall\, \lambda \in (0, r_u] \tag{5.7}$$

with non-zero probability.

Along a search direction, an SNN-GPP manifests as a directional derivative sign change from negative to positive. This definition makes allowance for the case where $\tilde{F}_n(\alpha^*) = 0$ may not exist at α^*, while also generalizing to the case where $\mathcal{F}(\alpha^*) = 0$ or $\bar{F}(\alpha) = 0$. In the latter two continuous cases, the probability of encountering an SNN-GPP is 1. We numerically demonstrate this assertion by estimating the probability density functions of encountering minima and SNN-GPPs along a search direction in the third column of Figure 5.3. We conduct 100 line searches with 100 increments of fixed step size $\alpha_{n,I_n} = 0.0007$ along the steepest descent direction for the Iris problem. We capture the frequency and location of minima and SNN-GPPs along the search direction and divide the frequency by the total number of occurrence to estimate the probability density function of minima and SNN-GPPs occurring. For continuous functions, all the probability of encountering a minimum or SNN-GPP is isolated to at $\alpha^* = \alpha_{snngpp}$. The same applies for static MBSS, with the only difference being that each mini-batch has an optimum at a different location. We call the spatial range of these separate mini-batch optima B_ϵ, which essentially defines a 1-D ball.

In the case of dynamic MBSS, the sampling induced discontinuities cause local minima to occur uniformly across the sampled domain. Importantly, the same does not apply to SNN-GPPs, which remain bounded within the range B_ϵ. This bound depends on the variance of $\tilde{F}'_n(\alpha)$ across mini-batches sampled using dynamic MBSS. In our foundational example, the range of B_ϵ has defined edges, as there are only 4 mini-batches, with distinct locations of SNN-GPPs between the step sizes $\alpha \in [0.025, 0.04]$. Outside of B_ϵ, the probability of encountering a sign change is zero for the given search direction, regardless of the order in which the mini-batches are selected. However, within B_ϵ sign changes can occur at varying probabilities, depending on how mini-batches are selected. An SNN-GPP will only be encountered when a previous mini-batch results in a negative directional derivative, and the next results in a positive directional derivative. This can only occur within the range of the first and last mini-batch's sign change along a search direction.

However, unlike in our foundational example, most practical implementations of dynamic MBSS in neural network training uniformly sample the content of mini-batches, $\mathcal{B}_{n,i}$, directly from the training set. In Figure 5.4 we sample $\mathcal{B}_{n,i}$ with different batch sizes, ranging from 10 samples to the full selected training set of 76 samples. The plots shown are the result of conducting 500 runs over $\alpha_{n,I_n} \in \{0, 0.002, 0.004, \ldots, 0.2\}$ and estimating the PDFs of locating local minima and SNN-GPPs along the steepest descent direction as obtained from the full-batch sampled loss, $\boldsymbol{d} = -\nabla\mathcal{L}(\boldsymbol{x}_n)$ at fixed starting point $\boldsymbol{x}_n$. However, here we show the distributions in the log domain for clearer comparison. For line search methods, a tightly bound distribution is desired, as it indicates the location of an optimum with high reliability. Here, the distribution of SNN-GPPs approximates a Gaussian form, centred around the true optimum and with increasing variance as $|\mathcal{B}_{n,i}|$ is reduced. Recent work suggests that the variance in sampled gradients is representative of α-distributions with $\alpha < 2$ [Simsekli et al., 2019], implying that the tails of the distributions are longer than those of Gaussians. The tails of α-distributions, like Gaussians, eventually tend towards 0, meaning that the likelihood of encountering an SNN-GPP diminishes far from the true optimum. However, this relationship does not hold for encountering local minima in dynamically MBSS losses, as shown in Figure 5.4(a). The distribution of candidate minimizers in Figure 5.4(a) approximates a uniform distribution over the sampled domain, and is largely independent of mini-batch size, which is consistent with our foundational problem in Figure 5.3. It is only for the full-batch loss functions, that the location of the local minimum is bounded.



(a) Function values

(b) Directional derivatives

Figure 5.4: Comparison between the estimated PDFs of (a) local minima and (b) SNN-GPPs along the steepest descent direction using uniformly sampled mini-batches, for the Iris dataset with mini-batch sizes ranging from $|\mathcal{B}_{n,i}| = 10$ to $M = 76$ (subscripts omitted for compactness). The x-axis indicates the step size α in 100 increments along $\boldsymbol{d} = -\nabla\mathcal{L}(\boldsymbol{x})$ from a fixed starting point. The y-axis shows the log of the probability of encountering a local minimum or SNN-GPP along the search direction. SNN-GPPs are more localized around the true optimum at ≈ 0.076 with in accuracy proportionate to sample size. Local minima are approximately uniformly distributed about the sampled domain for all sample sizes but M.

This experiment demonstrates the benefit of searching for SNN-GPPs when implementing dynamic MBSS rather than function minimizers. The location of SNN-GPPs is bounded, meaning that an estimate of the location of the true, full-batch minimum can be obtained. GOLS-I employs an inexact line search approach, which interacts well with the approximate nature of SNN-GPPs, while also reducing the computational cost of the method compared to exact line searches [Kafka and Wilke, 2019b]. The availability of GOLS-I to determine step sizes for dynamic MBSS loss functions leads to the question as to how the method interacts with existing popular neural network training algorithms.

5.5 Training algorithms and constructing search directions

Recall the algorithms selected for this investigation, namely: (Stochastic) Gradient Descent (SGD)[Morse and Feshbach, 1953], SGD with momentum (SGDM) [Rumelhart et al., 1988], Nesterov's Accelerated Gradient (NAG) [Nesterov, 1983], Adagrad [Duchi et al., 2011], Adadelta [Zeiler, 2012] and Adam [Kingma and Ba, 2015].

These algorithms can be divided into two groups:

1. Coupled directions algorithm class: SGD, SGDM and NAG, in which learning rates operate only on the length scale of search directions. The ratios between components in the search direction remain fixed, while the learning rate modifies the magnitude of the search direction vector. And,

2. Uncoupled directions algorithm class: Adagrad, Adadelta and Adam. These algorithms incorporate separate learning rate schedules along each individual component of the search direction, based on the respective component's historical gradient information. This results in the search directions that vary significantly in both magnitude and orientation from historical gradient vectors.

The coupled class encompasses SGD as well as alterations to its formulation to alleviate unwanted behaviour. The addition of momentum is an attempt to allow SGD the generation of mild ascent directions in order to overcome small local minima. It also alleviates the consecutively orthogonal direction behaviour of gradient descent [Arora, 2011] that would also apply to SGD, instead constructing directions that follow the contours of the loss function more closely. The NAG algorithm is a further improvement of SGD with momentum, incorporating a pre-emptive update step.

In contrast, the uncoupled class of algorithms applies a learning rate schedule along every component of the weight vector, which is dependent on the history of gradient magnitudes in each respective dimension. These algorithms have shown to be effective in generating search directions for the non-convex loss functions of deep learning tasks. Each algorithm in this group proposes a different schedule. Adagrad proposes an inverse relationship of the step size to the sum of squared gradient components. The result of this is that as the sum increases, the step size decreases, which can be problematic during training runs of many iterations, as updates along dimensions with large gradient components quickly tend to zero. Adadelta and Adam are two different formulations that seek to rectify this behaviour.

Importantly, all of the mentioned algorithms (except Adadelta) still have a learning rate related hyperparameter that needs to be determined. To integrate line searches into these algorithms, consider the formulation of the standard update step:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_{n,I_n}\boldsymbol{u}_n, \tag{5.8}$$

with the unit vector $\boldsymbol{u} = \frac{\boldsymbol{d}_n}{\|\boldsymbol{d}_n\|}$ dictating the unit search direction along which an update is to be made, while $\alpha_{n,I_n}\|\boldsymbol{u}_n\|$ determines the distance or step along the direction to be taken. This framework uncouples the direction of the update step, $\boldsymbol{u}_n$ from its magnitude, α_{n,I_n}, which can be useful for interpretation of the problem in the context of a line search. However, it is the unscaled direction $\boldsymbol{d}_n$ that is given by most learning algorithms. Evaluating the norm thereof is unnecessary, since a line search can compensate for the magnitude of $\boldsymbol{d}_n$ directly. As we are predominantly interested in determining the step size required for the given direction, we pose all given algorithms in the form:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_{n,I_n}\boldsymbol{d}_n. \tag{5.9}$$

Although some of the algorithms considered in this investigation were not originally proposed within this framework, we have taken the liberty to cast them into the form of Equation (5.9), as given in Appendix A.4. Since we are able to determine α_{n,I_n} for the various algorithms using a number of different line search methods [Kafka and Wilke, 2019b], we augment the name of the training algorithm with the prefix "LS-" in Appendix A.4 to accommodate the name of the line search. Therefore, as an example, implementing GOLS-I to determine α_{n,I_n} for LS-SGD is called "GOLS-I SGD". Note, that the original formulations of these algorithm can be recovered by using fixed step sizes for α_{n,I_n}. This concludes the ingredients required to train neural networks using gradient-only line searches with the search directions given by the selected algorithms.

5.6 Numerical studies

We consider the 7 chosen training algorithms combined with 23 different datasets, and in total 3 different types of network architecture, namely single hidden layer feed forward networks, deep feed forward networks and the ResNet18 convolutional network architecture. Dynamic MBSS is implemented for all investigations conducted in this chapter. We first consider performances of individual datasets and training algorithms, exploring the interaction between GOLS-I on a variety of problems. We compare GOLS-I to 3 fixed step sizes ranging over 3 orders of magnitude. Importantly, the fixed step sizes are manually chosen to demonstrate a range of training performances that are slow, efficient and unstable respectively. Conversely, GOLS-I is implemented without any user intervention and performance is compared to that of manually chosen learning rates, while the computational cost of selecting appropriate fixed learning rates is omitted in the comparison. Thereafter we explore the average performances of GOLS-I over the pool of datasets for the various algorithms.

Although single hidden layer neural networks have fallen out of favour over deep neural networks, they still offer significant research value to identify insight and isolate the essence concepts. Subsequently, we extend the investigation to deep networks of up to 6 hidden layers to demonstrate the robustness of GOLS-I in the context of the deep learning. Finally, we implement GOLS-I for training the ResNet18 architecture with 4 Convolutional layers on the well known CIFAR10 dataset.

5.6.1 Single hidden layer feedforward neural networks

We implement single hidden layer feedforward networks with the mean squared error (MSE) loss [Prechelt, 1994] to 22 classification datasets. The datasets span from 1936 to 2016 with sizes that vary from 150 to 70 000 observations. These are used to investigate the generality of combining GOLS-I with different training algorithms on datasets with diverse characteristics. The details of the chosen datasets and associated parameters with network architectures are summarized in Table 5.2. The number of nodes in the hidden layer for every dataset were determined by a combination of a proposed guideline [Yu, 1992] and a regression upper bound, discussed in Appendix A.5.

The selected fixed step sizes are split into small, medium and large respectively, each experimentally determined by conducting a number of training runs over a all datasets. The magnitudes were chosen such that the medium fixed step sizes would give the most competitive performance, while the small steps would cause slow convergence and the large steps would result in divergent behaviour. The range between small and large fixed step sizes were fixed at 3 orders of magnitude, in order to highlight the sensitivity of these parameters, see Table 5.1. However, the learning rates for the MNIST problem were more sensitive than the rest, such that the large fixed step was reduced to the point where the algorithms did not crash. We initialize GOLS-I with its minimum step size, $\alpha_{min} = 1 \cdot 10^{-8}$, in order to allow the line search automatically adapt the step size to the given problem. This avoids incorporating bias into the line search, which might suit a particular problem.

Once established, no adjustments were made to algorithms between application to different datasets. All datasets were prepared according to well known guidelines set out by Prechelt [1994] and divided into a training, test and validation dataset respectively by the ratios 2:1:1. The training set is used to optimize the model, while we consciously split the remaining data into validation and test datasets to show the generality of the training algorithms. Validation datasets are typically used to govern stopping criteria or tune other hyperparameters, while

	LS-SGD, LS-NAG	**LS-SGDM, LS-Adadelta**	**LS-Adagrad, LS-Adam**
Small Fixed Step	1	0.1	0.01
Medium Fixed Step	10	1	0.1
Large Fixed Step	100	10	1

Table 5.1: Learning rate ranges as used for different optimization algorithms.

	Dataset properties					**ANN properties**
No.	**Dataset name**	**Author**	**Observations,** M	**Inputs,** D	**Classes,** K	**Hidden nodes,** H
1	Cancer1	Prechelt [1994]	699	9	2	8
2	Card1	Prechelt [1994]	690	51	2	5
3	Diabetes1	Prechelt [1994]	768	8	2	7
4	Gene1	Prechelt [1994]	3175	120	3	9
5	Glass1	Prechelt [1994]	214	9	6	5
6	Heartc1	Prechelt [1994]	920	35	2	3
7	Horse1	Prechelt [1994]	364	58	3	2
8	Mushroom1	Prechelt [1994]	8124	125	2	22
9	Soybean1	Prechelt [1994]	683	35	19	3
10	Thyroid1	Prechelt [1994]	7200	21	3	20
11	Abalone	Nash et al. [1994]	4177	8	29	7
12	Iris	Fisher [1936]	150	4	3	3
13	H.A.R.	Anguita et al. [2012]	10299	561	6	7
14	Bankrupted Co. (yr. 1)	Zięba et al. [2016]	7027	64	2	35
15	Defaulted Credit Cards	Yeh and Lien [2009]	30000	24	2	23
16	Forests	Johnson et al. [2012]	523	27	4	6
17	FT Clave	Vurkaç [2011]	10800	16	4	15
18	Sensor-less Drive	Paschke et al. [2013]	58509	48	11	47
19	Wilt	Johnson et al. [2013]	4839	5	2	4
20	Biodegradable Compounds	Mansouri et al. [2013]	1054	41	2	8
21	Simulation Failure	Lucas et al. [2013]	540	20	2	8
22	MNIST Handwriting	Lecun et al. [1998]	70000	784	10	30

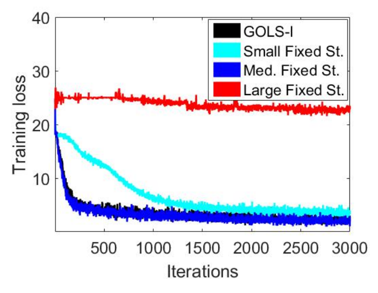
Table 5.2: Properties of the datasets considered for the investigation with the corresponding neural network architecture.

the test dataset is used to assess the final fitness of the network. Therefore, good correlation between validation and test dataset performances is desired, such that calibration on the validation dataset results in network performance that transfers well to the test dataset. Though we did not do any hyperparameter tuning, and our training runs were conducted by limiting the number of iterations, we include both validation and test dataset loss evaluations in our investigation to demonstrate that both validation and test datasets where large enough to fairly represent generalization performance. This avoids selecting biased ”generalization” data subsets, as most of the considered datasets do not have predetermined training, validation and test dataset separations.

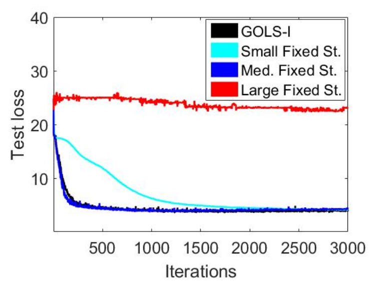
During training, the mini-batch sizes was kept constant at $|\mathcal{B}_{n,i}| = 32$, where each batch was uniformly selected with replacement for every evaluation of $\tilde{L}(\boldsymbol{x})$ and $\tilde{\boldsymbol{g}}(\boldsymbol{x})$ across all datasets. Training, validation and test dataset loss evaluations were averaged over 10 training runs of 3000 iterations each. Initial network configurations, $\boldsymbol{x}_0$, were randomly sampled from a uniform distribution with range $[-0.1, 0.1]$. These experiments were conducted using Matlab [Mathworks, 2015].

GOLS-I adapting to different datasets using LS-SGD

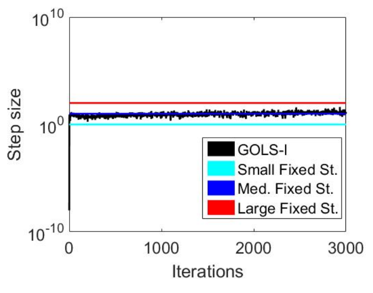
Firstly, we consider the training performance of a single algorithm with the selected fixed step sizes as well as GOLS-I. Hence we choose LS-SGD and train on the ”Forests” [Johnson et al., 2012, Chevallier and Jakubowska, 2018], Sensor-less Drive” [Paschke et al., 2013, Wenkel, 2018] and ”Simulation Failure” Lucas et al. [2013], Bischl et al. [2017] datasets. We show these particular datasets, as they are representative examples of performances observed over all datasets. The average training and test losses as well as the step sizes over 10 training runs are shown in Figure 5.5. The fixed step sizes represent the desired range, from slow to divergent training, while the medium step size is the best performer of the fixed step sizes. Note that the performance of GOLS-I is competitive or superior to that of the medium step size for the considered problems. However, the step sizes determined by GOLS-I have different characteristics for each example respectively.



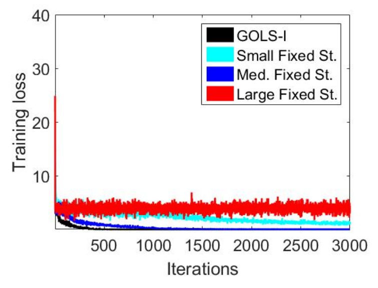
(a) Forest dataset



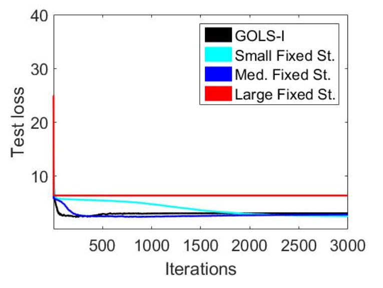
(b) Forest dataset



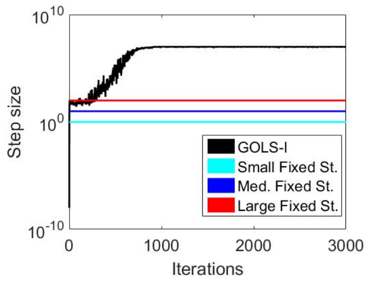
(c) Forest dataset



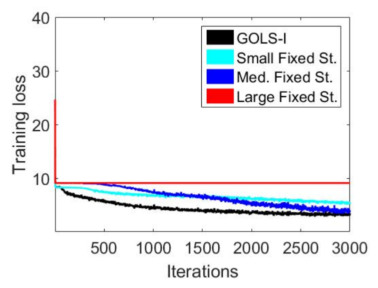
(d) Simulation Failure dataset



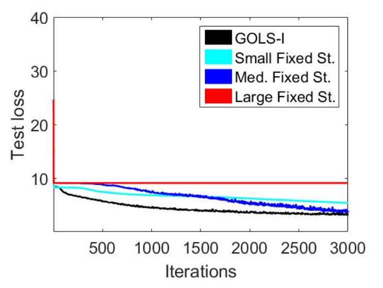
(e) Simulation Failure dataset



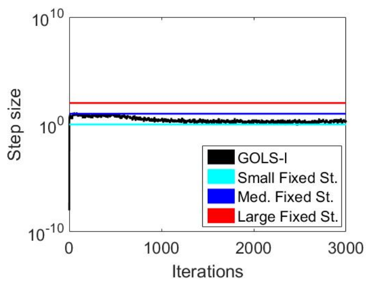
(f) Simulation Failure dataset



(g) Sensor-less Drive dataset



(h) Sensor-less Drive dataset



(i) Sensor-less Drive dataset

Figure 5.5: LS-SGD employed with fixed step sizes and GOLS-I, as applied to (a)-(c) the "Forests" [Johnson et al., 2012], (d)-(f) "Sensor-less Drive" [Paschke et al., 2013] and (g)-(i) "Simulation Failure" Lucas et al. [2013] datasets. GOLS-I successfully adapts step sizes according to the characteristics of the problem.

In the case of the Forests dataset in Figure 5.5(a)-(c), GOLS-I's determined a step sizes closely approximate the medium fixed step. This was the most commonly observed trend, with 17 out of 22 datasets demonstrating step sizes close to that of the medium step, or starting at the medium step size and slowly tending towards the larger step size as training progresses. Recall, that the initial guess for the step size in the first iteration of GOLS-I is a very conservative $\alpha_{0,0} = 10^{-8}$. The given results show that GOLS-I adapts the step size from its conservative initial guess to the same magnitude as a manually chosen competitive fixed step size from the first iteration.

There are other instances, where GOLS-I continues to grow step sizes until the upper limit of allowed step sizes is reached. One out of 3 such cases is the Simulation Failure dataset in Figure 5.5(d)-(f). The performance of GOLS-I is marginally superior to that of the medium fixed step, while determining step sizes that are closer to the large fixed step in the beginning of training. However, due to the adaptive nature of GOLS-I (possibly influenced by its ramping up from a small initial guess), it does not diverge like the large fixed step. As training progresses for this problem, and the algorithm approaches an optimum, the gradient magnitude diminishes, which causes GOLS-I to increase the step sizes in search of an SNN-GPP. This is a consequence of a decreasing gradient magnitude and the update step formulation proposed in Equation (5.9).

The performance of GOLS-I for the Sensor-less Drive dataset in Figure 5.5 is particularly interesting. Here, the step sizes determined by GOLS-I start close to the medium fixed step and decrease gradually to the small step size. This occurred for 2 out of the 22 considered problems. In this case, the training performance of GOLS-I is superior to both the medium and the small step sizes. This offers confirmation that GOLS-I is capable of automatically adapting the step size to the requirements of the problem. Conversely, it would be infeasible to pre-empt the shape of the learning rate schedules, such as determined by GOLS-I, for each of the shown problems.

GOLS-I adapting to different algorithms of a fixed dataset

Subsequently, we restrict the investigation to the "Forests" [Johnson et al., 2012] dataset, while considering the LS-NAG, LS-Adadelta and LS-Adam algorithms with their respective fixed step sizes and GOLS-I implementations. The training and test dataset losses with corresponding step sizes are given in Figure 5.6. GOLS-I NAG performs similarly to the case considered for the same dataset as applied to GOLS-I SGD, see Figure 5.5(a)-(c). This is the most common case, where GOLS-I determines step sizes close to the medium fixed step during the early stages of training, while tending towards the large fixed step towards the latter half. Here, the training performance of GOLS-I is indistinguishable from that of the medium fixed step.

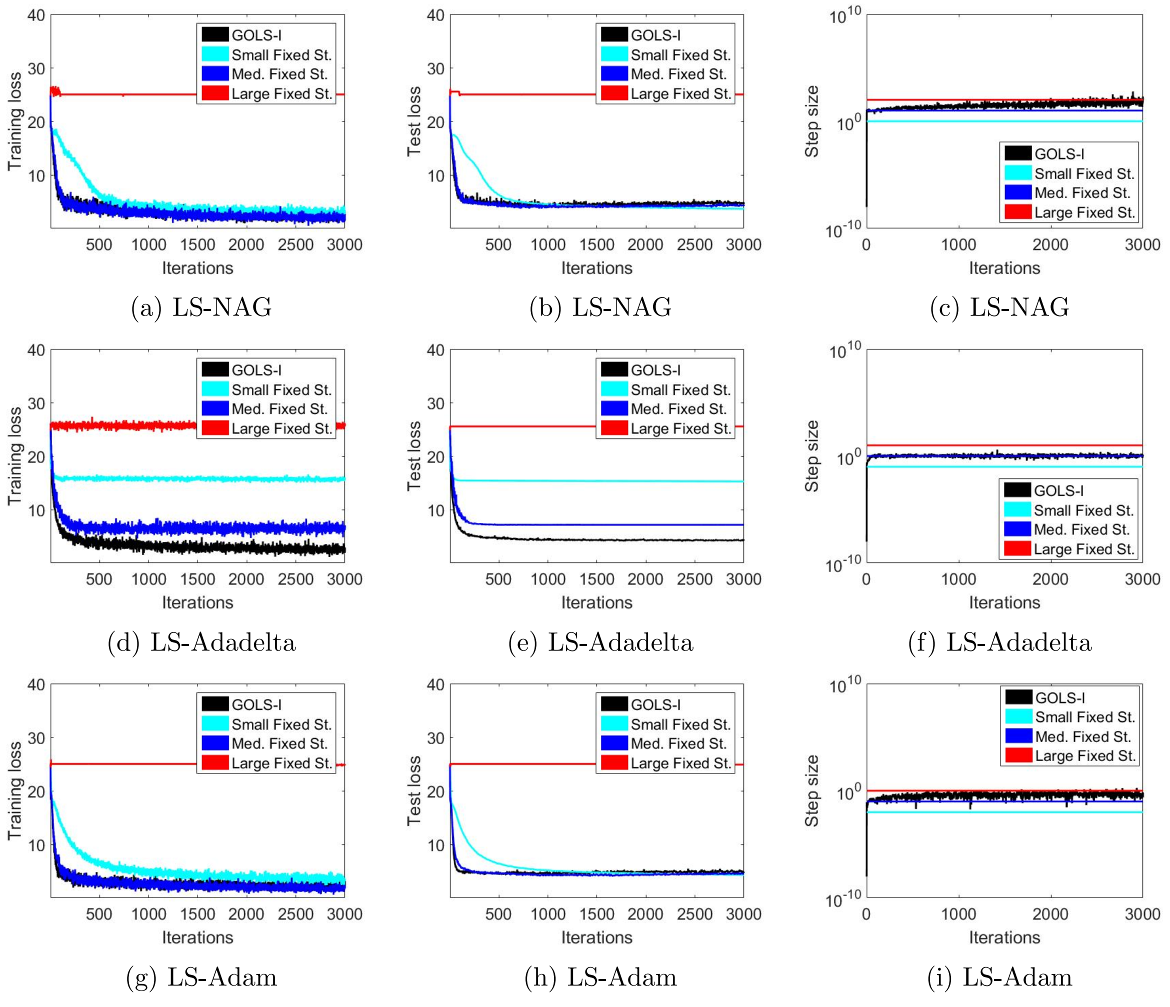


Figure 5.6: The "Forests" [Johnson et al., 2012] problem as trained using (a)-(c) LS-NAG, (d)-(f) LS-Adadelta and (g)-(i) LS-Adam with both GOLS-I and fixed step sizes. GOLS-I automatically recovers step sizes close to medium fixed steps for the shown algorithms, while each algorithm has different medium fixed step sizes, each an order of magnitude apart.

The interaction between GOLS-I and LS-Adadelta is noteworthy. Adadelta on its own does not contain a learning rate parameter. However, we introduced the scalar modifier, $\alpha_{n,I_n} = 1$, to its formulation in order to explore the characteristics of its search directions. Unsurprisingly, LS-Adadelta's best performance with fixed step sizes is when $\alpha_{n,I_n} = 1$, i.e. recovering its original formulation. However, the improvement in performance of GOLS-I Adadelta is significant: The GOLS-I recovers average step sizes close to 1, while its performance is superior to that of fixed step size 1 in both training and test losses. This implies that the small deviations from the medium step sizes according to updates performed with GOLS-I improve the adaptation of LS-Adadelta to the characteristics of the encountered loss surface.

GOLS-I Adam returns similar performance characteristics to those of GOLS-I NAG, with the step size increasing from the medium step size as training progresses. However, the medium fixed step sizes for LS-NAG, LS-Adadelta and LS-Adam are all an order of magnitude apart at 10, 1 and 0.1 respectively. This means that GOLS-I is able to automatically adapt the determined step size according to the characteristics of the training algorithm dictating the search direction, while resulting in superior or competitive training performance.

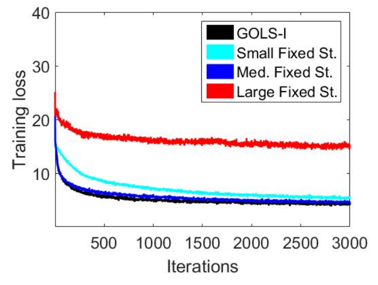
Averaged comparison of algorithms over 22 datasets

Now that it has been established that GOLS-I fulfils its intended purpose for independent datasets and algorithms, we consider the average performance of all 6 algorithms over the 22 considered problems in order to examine the generality of the method. The resulting averaged training, validation and test dataset losses are shown in Figures 5.7 and 5.8. As demonstrated, the fixed step sizes encompass the performance range from too slow, to divergent within 3 orders of magnitude. For our implementation of LS-Adagrad, the largest fixed step does not diverge, but rather results in worst generalization of the 3 step sizes.

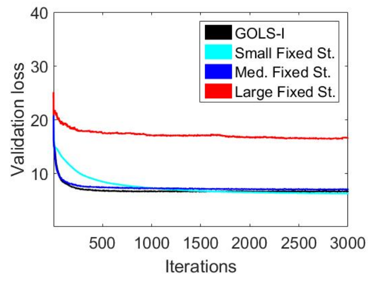
For most algorithms with the use of GOLS-I, the average validation and test set losses show that overfitting occurs after approximately 1000 iterations for the datasets considered in this analysis. Importantly, the characteristics of the validation and test loss plots are comparable. This means that the network designer can reliably use the behaviour of the validation dataset to enforce a stopping criterion or tune other network parameters.

As demonstrated in Sections 5.6.1 and 5.6.1, the training loss for GOLS-I is mostly either the best performer or competitive with the medium fixed step. The obvious outlier is LS-SGDM, where GOLS-I performs worse than any fixed step. This is a result of LS-SGDM adding momentum, which is a fixed fraction of the previous update after the line search has already determined the step size. This forces the optimizer past minima and up ascent directions. For this reason, LS-SGDM is ill-suited to be used with a line search such as GOLS-I. Conversely, small, fixed step sizes result in update steps with incremental changes in the loss function, which also translates to less aggressive input from the momentum term, allowing the algorithm to more closely follow the contours of the loss function. The ascending behaviour of GOLS-I SGDM can be addressed by reducing the parameter γ_m, shifting the algorithm's behaviour closer to LS-SGD. The adaptation to the momentum algorithm offered by LS-NAG is effective in overcoming the ascending problem, resulting in performance comparable to LS-SGD for the coupled directions class of algorithms.

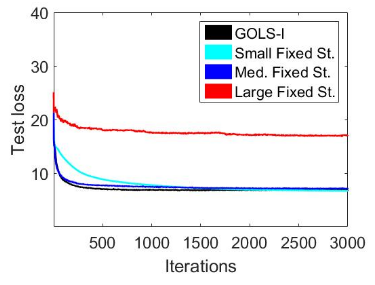
In terms of the uncoupled class of algorithms, GOLS-I performed best with LS-Adagrad and LS-Adadelta relative to fixed step sizes. GOLS-I Adagrad produced an average performance that was superior to that of the medium fixed step, while GOLS-I Adadelta on average performed similarly to the results explored in Section 5.6.1. As discussed, this is an indication that the bottleneck for LS-Adadelta is not the search direction, but rather the step size along its search direction. LS-Adam in this case is the outlier, with the smallest fixed step performing best as training extends beyond approx 1000 iterations. This is due to LS-Adam's formulation, which includes a term that acts like momentum. However, this effect is investigated more clearly in Section 5.6.2 considering deep networks.



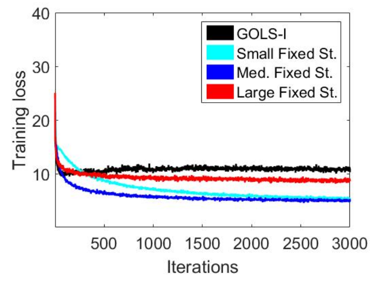
(a) LS-SGD



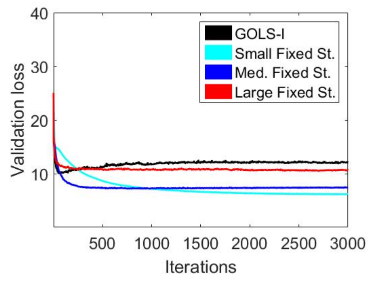
(b) LS-SGD



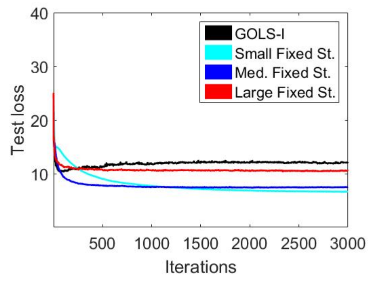
(c) LS-SGD



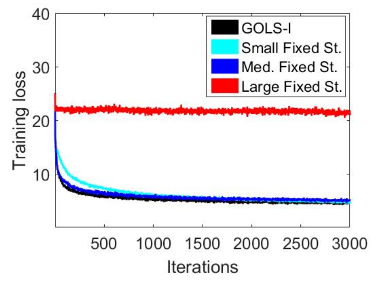
(d) LS-SGDM



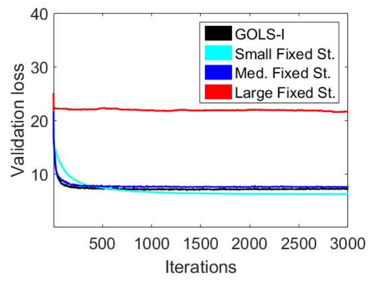
(e) LS-SGDM



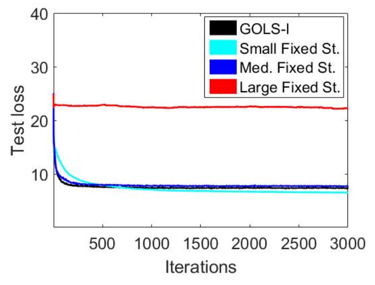
(f) LS-SGDM



(g) LS-NAG



(h) LS-NAG



(i) LS-NAG

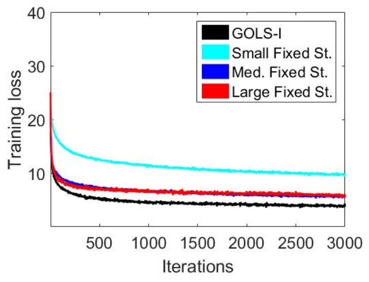
Figure 5.7: The training, test and validation dataset errors for (a)-(c) LS-SGD, (d)-(f) LS-SGD with momentum and (g)-(i) LS-NAG, averaged over all datasets for 10 different starting points per analysis. The algorithms are implemented with different fixed step sizes and the gradient-only based GOLS-I.

Average performance over all algorithms

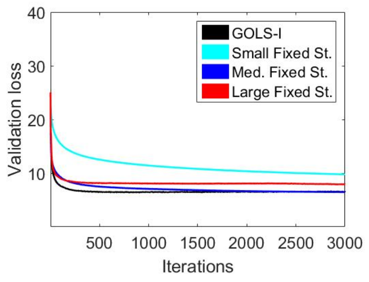
We consider the average performance over all datasets and all algorithms in Figure 5.9. We do this in order to highlight two aspects: 1) How GOLS-I generalizes over all considered dataset-algorithm combinations and 2) How GOLS-I competes overall in terms of function evaluations per iteration compared to fixed step sizes.

The results up to and including Figures 5.9(a)-(c) are given in terms of iterations, as the emphasis is on the comparison of performance gained and step size determined per iteration between GOLS-I and the fixed steps. Here GOLS-I is competitive with or superior to the selected medium step size.

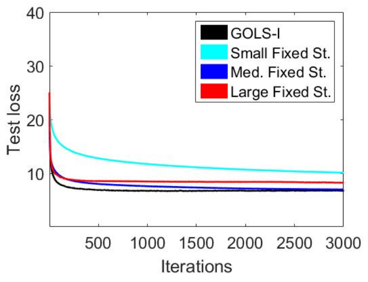
As a line search, GOLS-I performs a number of function evaluations per iteration. For the purposes of this chapter, a function evaluation is considered to be an evaluation of $\tilde{\boldsymbol{g}}(\boldsymbol{x})$. A constant step size uses only a single function evaluation per iteration. On average GOLS-I performed between 1-4 function evaluations per iteration, depending on the algorithm it was coupled to, as well as the characteristics of the loss functions during training. Thus to compare the relative computational cost of GOLS-I compared to fixed step sizes, the average algorithm performances of the constant step methods and GOLS-I are plotted in terms of function evaluations in Figures 5.9(d)-(f). Although GOLS-I is on average a factor of 2.5 more expensive compared to a well chosen medium step size, the extra cost does not offset the benefit gained by not requiring to search for an effective fixed step size.



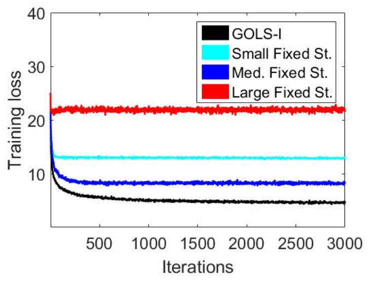
(a) LS-Adagrad



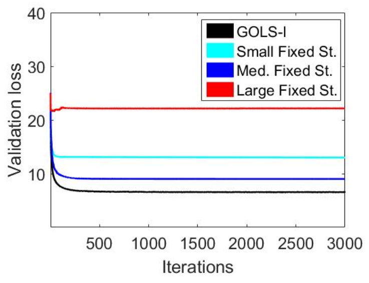
(b) LS-Adagrad



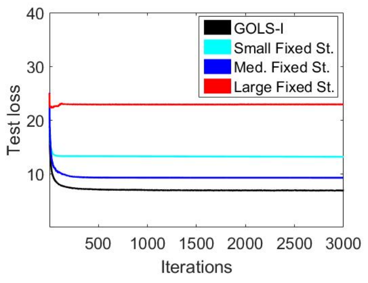
(c) LS-Adagrad



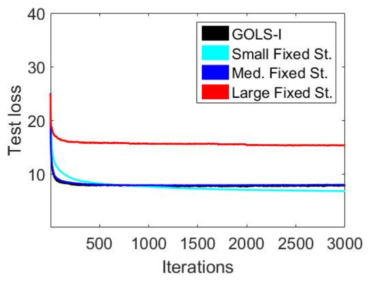
(d) LS-Adadelta



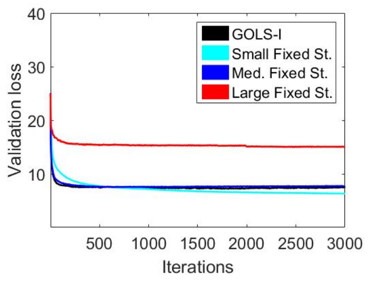
(e) LS-Adadelta



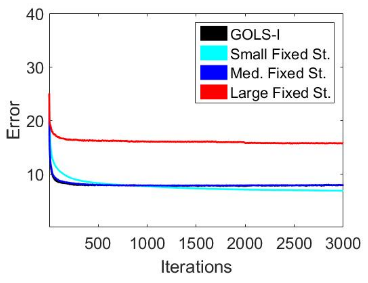
(f) LS-Adadelta



(g) LS-Adam



(h) LS-Adam



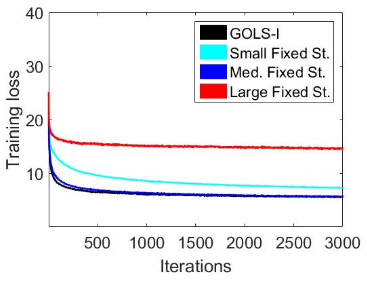
(i) LS-Adam

Figure 5.8: The training, test and validation dataset errors for (a)-(c) LS-Adagrad, (d)-(f) LS-Adadelta and (g)-(i) LS-Adam, averaged over all datasets for 10 different starting points per analysis. The algorithms are implemented with different fixed step sizes and the gradient-only based GOLS-I.

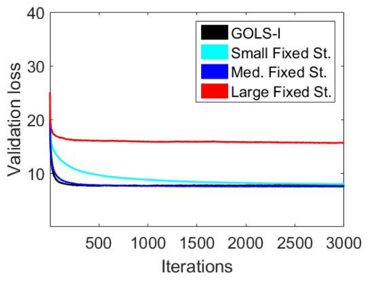
5.6.2 Multiple hidden layer neural networks

Next, we extend our analyses to include deep neural networks. We consider GOLS-I with LS-SGD, LS-Adagrad and LS-Adam; each applied to neural networks with a increasing numbers of hidden layers [Bengio, 2009], ranging from 1 to 6. All parameters remained consistent with those of the analysis described in Section 5.6.1, with exception of the number of hidden layers. This investigation omitted the larger datasets, namely: Simulation failures [Lucas et al., 2013], Defaulted credit cards [Yeh and Lien, 2009], Sensor-less drive diagnosis [Paschke et al., 2013] and MNIST [Lecun et al., 1998], which makes a total number of datasets 18 for this section.

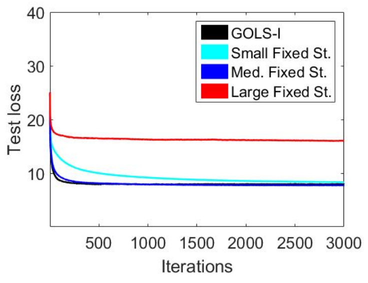
Figures 5.10(a)-(c) show the average training, validation and test losses over 1 to 6 hidden layers using GOLS-I SGD. Here, variance clouds are shown with shaded colours around the mean, which is denoted by a solid line. Reasonable training performance can be achieved for smaller networks of 1 to 3 hidden layers in 3000 iterations. Convergence is slow on networks with 4 hidden layers, although some progress is made within 3000 iterations. However, there is a severe decrease in performance for networks with 5 and 6 hidden layers, with higher variance between training runs, compared to smaller networks. This indicates, that the diminishing gradient problem [Hochreiter and Frasconi, 2009] begins to affect the optimization algorithm considerably from this network depth onwards. This plot is representative not only for GOLS-I SGD, but also for analyses not shown here concerning GOLS-I SGDM and the GOLS-I NAG algorithms, indicating the difficulty present for the coupled directions class of algorithms for this



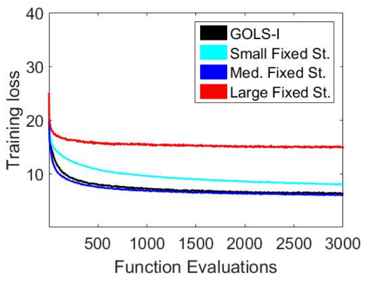
(a)



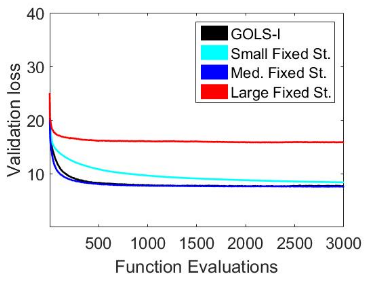
(b)



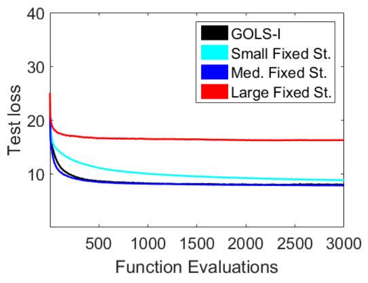
(c)



(d)



(e)



(f)

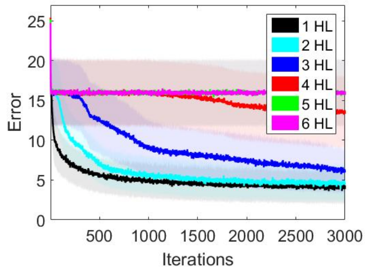
Figure 5.9: Average training, test and validation dataset errors over all datasets and all algorithm in terms of (a)-(c) iterations and (d)-(f) function evaluations. For GOLS-I, the number of function evaluations per iteration is on average 2.5, whereas that of a fixed step size approach is always 1. This is also a proxy for computational cost. Therefore, plotting in terms of function evaluations corrects for this disparity. Though GOLS-I is a factor of 2.5 more expensive than the medium fixed step on average, it remains competitive, while requiring no tuning.

problem. It is also to be noted, that this problem can be alleviated with more advanced network initialization methods, as proposed by Glorot and Bengio [2010].

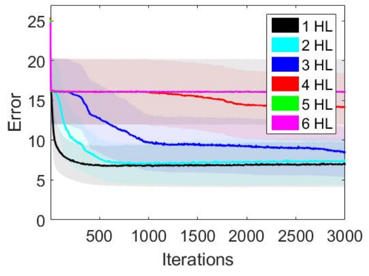
The corresponding results with the use of GOLS-I Adagrad are shown in Figures 5.10(d)-(f). As a representative of the uncoupled directions class of algorithms, GOLS-I Adagrad performs much more consistently across increasing numbers of hidden layers, exhibiting lower variance between training runs. However, there is still a slight relative decrease in performance for deeper networks. The test and validation error plots in Figures 5.10(e) and (f) show that validation and test dataset losses increase for the smaller networks due to overfitting, while the larger networks still progress towards a minimum, albeit at a slower convergence rate.

In the case of GOLS-I Adam, as shown in Figures 5.10(g)-(i), convergence occurs at a slower rate than with GOLS-I Adagrad for deeper networks. It is here, that the momentum-like term in LS-Adam hampers its performance with GOLS-I considerably. This term, $\hat{\boldsymbol{m}}_{n+1}$ in the numerator of its formulation, see Appendix A.4.6, is governed by the β_1 parameter. By default this parameter is set as $\beta_1 = 0.9$. However, if this parameter is reduced to $\beta_1 = 0$, the momentum behaviour of LS-Adam is deactivated, while the denominator still contributes to produce uncoupled, scaled directions. In Figures 5.10(j)-(l), the results of separate training runs of GOLS-I Adam with $\beta_1 = 0$ are shown. It is clear, that there is a vast improvement in performance, with performance now being comparable to that of GOLS-I Adagrad.

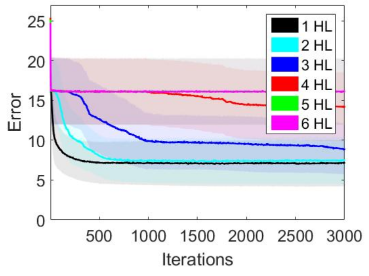
This investigation has confirmed that generally, momentum in the context of GOLS-I should be avoided. And secondly, that the element-wise scaling of the direction entries creates a total direction which allows for greater perturbation in the deeper layers of the networks, promoting faster training. The inclusion of GOLS-I into the considered training algorithm capitalizes on their various search directions for efficient, deep network training, while requiring no parameter tuning.



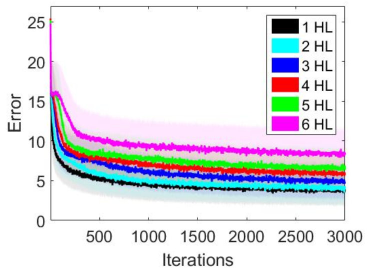
(a) GOLS-I SGD



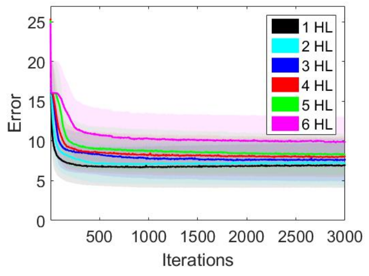
(b) GOLS-I SGD



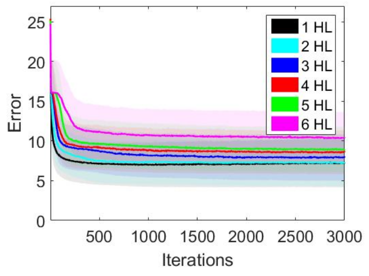
(c) GOLS-I SGD



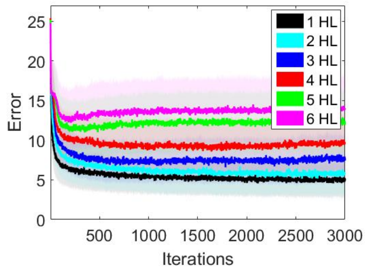
(d) GOLS-I Adagrad



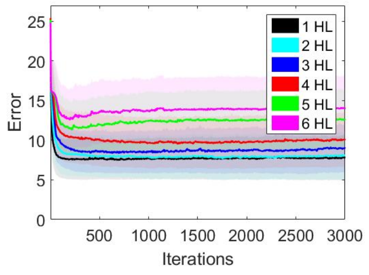
(e) GOLS-I Adagrad



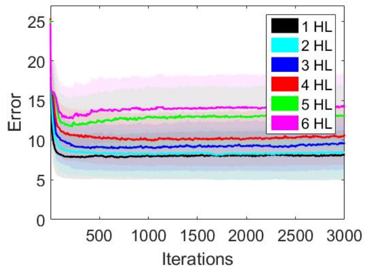
(f) GOLS-I Adagrad



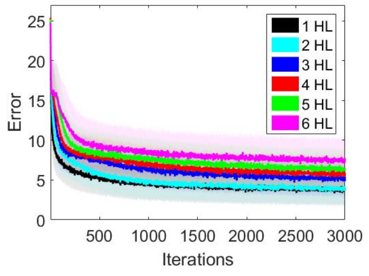
(g) GOLS-I Adam



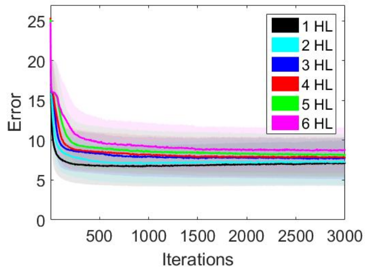
(h) GOLS-I Adam



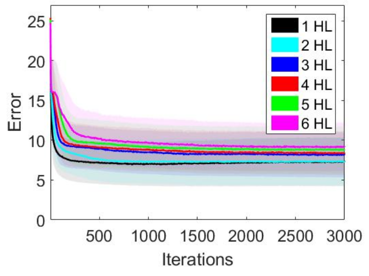
(i) GOLS-I Adam



(j) GOLS-I Adam, $\beta_1 = 0$



(k) GOLS-I Adam, $\beta_1 = 0$

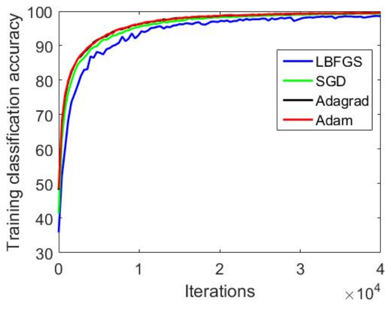


(l) GOLS-I Adam, $\beta_1 = 0$

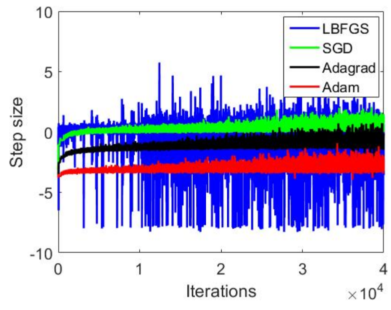
Figure 5.10: Training, test and validation dataset losses of (a)-(c) GOLS-I SGD, (d)-(f) GOLS-I Adagrad, (g)-(i) GOLS-I Adam and (j)-(l) GOLS-I Adam with $\beta_1 = 0$ for 1-6 of hidden layers. The averages are taken over a subset of datasets with 10 separate training runs per dataset.

5.6.3 CIFAR10 with ResNet18

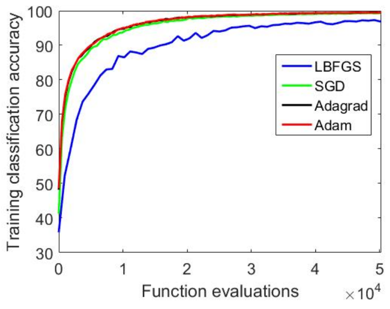
Now that GOLS-I has been demonstrated on a variety of foundational problems, we apply it to an architecture more likely to be encountered in practice. We therefore consider the CIFAR10 classification dataset with the ResNet18 Convolutional Neural Network (CNN) architecture [He et al., 2016]. The baseline PyTorch code for this problem was sourced from Liu [2018], which was then modified to accommodate GOLS-I. This experiment makes use of the cross entropy loss for $\ell(\boldsymbol{x}, \boldsymbol{t}_b)$, which is more popular for classification tasks, as it can be viewed as minimizing the KL divergence between the distributions of the neural network, and the target outputs [Goodfellow et al., 2016]. The algorithms considered with GOLS-I are LS-SGD, LBFGS [Nocedal, 1980] (as used with a line search as "LS-LBFGS"), LS-Adagrad and LS-Adam as adapted for PyTorch 1.0. Incorporating lessons learnt from Section 5.6.2, the momentum term in LS-Adam is turned off, i.e. $\beta_1 = 0$. The batch size chosen for this problem was constant at 128 and training was limited to 40,000 iterations. In this case, $\mathcal{B}_{n,i}$ is randomly sampled from the pool of training data without replacement, until all training observations have been sampled. Thereafter, the training dataset is shuffled and the process repeats.



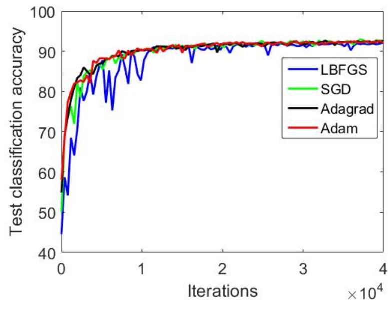
(a) Training classification accuracy



(b) Step sizes

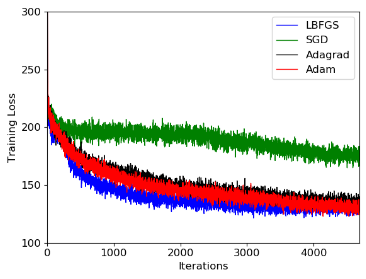


(c) Performance versus cost

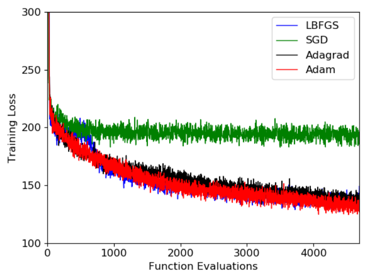


(d) Test classification accuracy

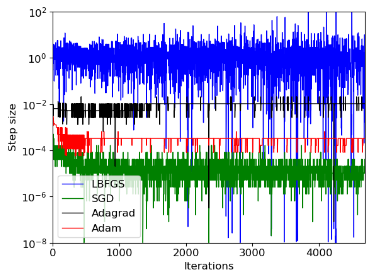
Figure 5.11: The (a) training classification accuracy, (b) step sizes, (c) training classification accuracy in terms of function evaluations and (d) test classification accuracy for various training algorithms as applied to the CIFAR10 dataset with the ResNet18 architecture. The step sizes are determined by GOLS-I, while the explicit reference to ”GOLS-I” is omitted in the legends for compactness. explicit The step sizes are different between algorithms, while their performances per iteration are comparable. This indicates that step sizes and not search directions may be the bottleneck for training this problem. However, the cost of each algorithm is not the same, revealing GOLS-I LBFGS to be the worst performer in terms of computational efficiency.



(a) Training loss



(b) Training loss



(c) Step sizes

Figure 5.12: The (a) training loss in terms of iteration, (b) training loss in terms of function evaluations, (c) step sizes for various training algorithms as applied to variational autoencoder [Zuo and Chintala, 2018] training. The step sizes as determined by GOLS-I are again different for every algorithm (with ”GOLS-I” omitted in the legends for compactness), while highlighting the effectiveness of Hessian approximations generated by LS-LBFGS.

The resulting training and test classification accuracy, step sizes and training classification in terms of function evaluations are shown in Figure 5.11. The performances of GOLS-I SGD, GOLS-I Adagrad and GOLS-I Adam are almost indistinguishable in training, while marginally outperforming GOLS-I LBFGS. This shows that for this problem, the quality of search directions constructed by GOLS-I LBFGS are inferior to those of the remaining algorithms. The GOLS-I LBFGS directions are generated based Hessian approximations, computed from the gradient information at previous iterations. However, the combination of making use of dynamic MBSS as well as GOLS-I being an inexact line search, reduces the quality of the hessian approximation and can result in the generation of ascent directions after a number of iterations. In such cases GOLS-I compensates by taking the minimum step size in that direction, resulting in the high variance in step sizes we observe in Figure 5.11(b). This is also confirmed by the relatively noisy training classification errors of GOLS-I LBFGS compared to the remaining algorithms.

We capture the number of function evaluations performed for each iteration and determine the average number of function evaluations per iteration during training, as summarized in Table 5.3. This shows, that GOLS-I LBFGS is on average 77% more expensive in the number of function evaluations performed per iteration than the remaining algorithms. To compare performance versus computational cost between the algorithms, we plot the training classification accuracy in terms of function evaluations in Figure 5.11(c). This plot highlights, that the search directions constructed by GOLS-I LBFGS do not offset the cost of their construction for this problem. However, given that this is to our knowledge the first GOLS-I LBFGS formulation that incorporates dynamic MBSS line searches in the construction of its hessian approximations without any further modifications, the offered performance is not to be discounted. Future work might consider, whether modifications such as increasing batch size or altering parameters in LS-LBFGS can improve its efficiency. However, the insignificant difference in performance between LS-Adagrad, LS-Adam and LS-SGD indicates, that the training algorithms are relatively insensitive to the inclusion of curvature information for this problem. Therefore, CIFAR10 with ResNet18 may not be a fitting platform to fully test the validity of LS-LBFGS' Hessian approximations in neural network training.

	LS-LBFGS	**LS-SGD**	**LS-Adagrad**	**LS-Adam, $\beta_1 = 0$**
Average # function evaluations per update	2.35	1.26	1.28	1.28

Table 5.3: Algorithmic comparison in terms of cost. A constant number of iteration was used to simplify the comparison of training performance. However, not all algorithms use the same number of function evaluations per iteration with GOLS-I coupled to different algorithms. Each function evaluation consists of calling only the gradient, $\tilde{\boldsymbol{g}}(\boldsymbol{x})$.

In the case of GOLS-I SGD, GOLS-I Adagrad and GOLS-I Adam, the step sizes quickly stabilize around a particular order of magnitude during training. As observed in Section 5.6.1, this magnitude differs for every algorithm, yet is automatically recovered by GOLS-I. The step sizes remain relatively constant, with oscillations of up to half an order of magnitude. Consistent step sizes significantly reduce the total cost of training. As training continues, these oscillations can increase to 4 orders of magnitude. We postulate that this increasing trend occurs due to mini-batch samples that contain observations, that are far from the decision boundary, resulting in small gradients; and mini-batches that are close to the decision boundary, resulting in large gradients. As GOLS-I adapts, it randomly alternates between these cases causes increasing the step sizes for small gradients, and the converse for larger gradients. To compensating for this behaviour, Smith et al. [2017] proposes gradually increasing the batch size during training.

	LS-LBFGS	**LS-SGD**	**LS-Adagrad**	**LS-Adam, $\beta_1 = 0$**
Accuracy in %	92.11	92.97	92.63	92.58

Table 5.4: Maximum achieved test classification accuracy after 40,000 training iterations on the CIFAR10 dataset with various training algorithms in combination with GOLS-I.

All algorithms considered in this investigation trained the model to a test classification accuracy of above 90%, which is evident from the plateau in test classification, see Figure 5.11(d). Although the search directions vary in formulation for different algorithms, the outcome is comparable with the use of GOLS-I to determine step sizes. The maximum test classification accuracy achieved by each algorithm is shown in Table 5.4. This investigation demonstrated, that GOLS-I is also effective in larger, practical problems. Provided that no momentum terms are present in the algorithms, vastly different algorithms can have comparable training performance with the use of GOLS-I. This is in contrast to an investigation by Mukkamala and Hein [2017], which used *a priori* selected learning rates and learning rate schedules to solve the same problem. The authors' study showed a significant variance in training error between different learning rate schedules; and reported a best test classification accuracy of 86% after the equivalent of $156,000$ function evaluations. An equivalent result was achieved in our investigation after $\pm 7,000$ function evaluations for GOLS-I SGD, GOLS-I Adagrad and GOLS-I Adam. Test classification accuracies achieved in our investigation are consistent with other studies [He et al., 2016, Huang et al., 2018], which implement manually tuned optimizers. He et al. [2016] make use of LS-SGDM with a $\alpha_{n,I_n} = 0.1$, which is reduced by an order of magnitude at *a priori* determined intervals, for a total of 64,000 training iterations with a mini-batch size of $|\mathcal{B}_{n,i}| = 128$, resulting in a test error of around 91%. Huang et al. [2018] employ LS-Adam with $\alpha_{n,I_n} = 0.01$, using a larger mini-batch size of $|\mathcal{B}_{n,i}| = 512$ to achieve a test classification accuracy of 93.52%. We achieve an average test classification error of 92.6% over GOLS-I SGD, GOLS-I LBFGS, GOLS-I Adagrad and GOLS-I Adam ($\beta_1 = 0$) with a mini-batch size of $|\mathcal{B}_{n,i}| = 128$, confirming that GOLS-I determined the step sizes of selected algorithms effectively, without requiring any prior parameter tuning.

5.7 Conclusion

This chapter demonstrated that stochastic gradient-only line searches that are inexact (GOLS-I) allow for a generalized strategy to determine the learning rates of different training algorithms as opposed to conducting extensive hyper-parameter tuning studies. We considered 23 datasets with sizes varying between 150 and 70 000 observations, input dimensions ranging between 4 and 3072 and output dimensions between 2 and 29. Network architectures considered were feed forward networks with 1 to 6 hidden layers of varying sizes and a convolutional architecture with residual connections. In addition a total of seven training algorithms namely: Gradient Descent, Gradient Descent with momentum, Nesterov's Accelerated Gradient Descent, Adagrad, Adadelta, Adam and LBFGS were considered. Our investigations have shown, that algorithms with momentum terms do not perform well with GOLS-I. However, if it is possible to remove the momentum component from the algorithm, performance with GOLS-I drastically improves, which was demonstrated on both shallow and deep feedforward neural networks. We also showed that without any parameter tuning, the performance of GOLS-I with LS-SGD, LS-BFGS, LS-Adagrad, and LS-Adam is competitive with other studies, which implement manually selected learning rate schedules.

The use of GOLS-I avoids the need for manual parameter tuning or computationally expensive global parameter optimization approaches to determine an effective average fixed learning rate or explore learning rate schedule parameters. Gradient-only line search strategies on average require more than one gradient evaluation per iteration, making them more expensive than fixed step size approaches per iteration. However, GOLS-I remains to our knowledge most effective adaptive learning rate optimization method to date. Additionally, our investigations have shown, that incorporating GOLS-I into the LBFGS algorithm shows promise for future improvements. This re-opens questions concerning the feasibility of conjugate gradient or Quasi-Newton methods in stochastic environments such as neural network training.

Chapter 6

The compatibility of Gradient-only line searches with various activation functions

Gradient-Only Line Searches (GOLS) can be used to determine step sizes in the discontinuous loss functions of dynamic mini-batch sub-sampled neural network training. Step sizes in GOLS are determined by localizing Stochastic Non-Negative Associated Gradient Projection Points (SNN-GPPs) along a descent direction, which are identified by a sign change in the directional derivative from negative to positive. Activation functions are a significant component of neural network architectures, as their smoothness and continuity characteristics directly effect the gradient characteristics of the loss function. Therefore, it is of interest to investigate the relationship between activation functions and different neural network architectures in the context of GOLS. We find that GOLS are robust for a range of activation functions, but sensitive to the Rectified Linear Unit (ReLU) activation function in standard feedforward architectures. The zero-derivative in ReLU's negative input domain can lead to the gradient-vector becoming sparse, which severely affects training. We show that implementing architectural features such as batch normalization and skip connections can alleviate these difficulties and benefit training with GOLS for all activation functions considered.

6.1 Introduction

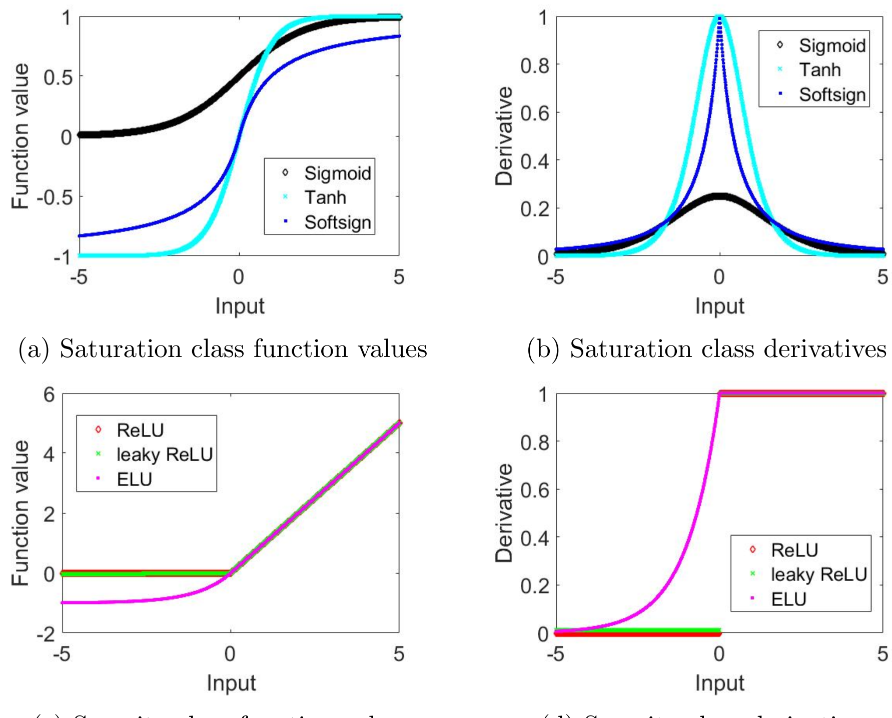
The introduction of gradient-only line searches (GOLS) [Kafka and Wilke, 2019b] has allowed learning rates to be automatically determined in the discontinuous loss functions of neural networks training with dynamic mini-batch sub-sampling (MBSS). The discontinuous nature of the dynamic MBSS loss is a direct result of successively sampling different mini-batches from the training data at every function evaluation, introducing a *sampling error* [Kafka and Wilke, 2019b]. To determine step sizes, GOLS locates *Stochastic Non-Negative Associated Gradient Projection Points* (SNN-GPPs), manifesting as sign changes from negative to positive in the directional derivative along a descent direction. This allows GOLS to strike a balance between the benefits of training using dynamic MBSS, such as 1) increasing the training algorithm's exposure to training data [Bottou, 2010] as well as 2) overcoming local minima [Saxe et al., 2013, Dauphin et al., 2014, Goodfellow et al., 2015, Choromanska et al., 2015]; and the ability to localize optima in discontinuous loss functions [Kafka and Wilke, 2019a]. This is in contrast to minimization line searches [Arora, 2011], which find false local minima induced by sampling error discontinuities. These have found to be uniformly spread along the descent direction, rendering minimization line searches ineffective for determining representative step sizes [Kafka and Wilke, 2019a,b].

Previous work has shown, that the *Gradient-Only Line Search that is Inexact* (GOLS-I) is capable of determining step sizes for training algorithms beyond stochastic gradient descent (SGD) [Robbins and Monro, 1951], such as Adagrad [Duchi et al., 2011], which incorporates approximate second order information [Kafka and Wilke, 2019b]. GOLS-I has also been demonstrated to outperform probabilistic line searches [Mahsereci and Hennig, 2017], provided mini-batch sizes are not too small (< 50 for investigated problems) [Kafka and Wilke, 2019]. The gradient-only optimization paradigm has recently also shown promise in the construction of approximation models to conduct line searches [Chae and Wilke, 2019].

Some of the most important factors governing the nature of the computed gradients are: 1) The neural network architecture, 2) the activation functions used within the architecture, 3) the loss function implemented, and 4) the mini-batch size used to evaluate the loss, to name a few. In this chapter, we concentrate on the influence of activation functions (AFs) on training performance of GOLS for different neural network architectures. Activation functions (AFs) have a direct influence on how a neural network processes incoming data, and by extension, dictate the nature of the gradients used in GOLS. We also consider the effect of architectural features such as batch normalization [Ioffe and Szegedy, 2015] and skip connections [He et al., 2016] on training architectures with different AFs using GOLS.

The AFs considered in this investigation can be split primarily into two categories, namely:

1. The saturation class [Xu et al., 2016]: Including Sigmoid [Han and Morag, 1995], Tanh [Bergstra et al., 2009] and Softsign [Karlik, 2015]; and
2. The sparsity class: Including ReLU [Glorot and Bordes, 2011], leaky ReLU [Maas et al., 2013] and ELU [Clevert et al., 2016].



(a) Saturation class function values (b) Saturation class derivatives

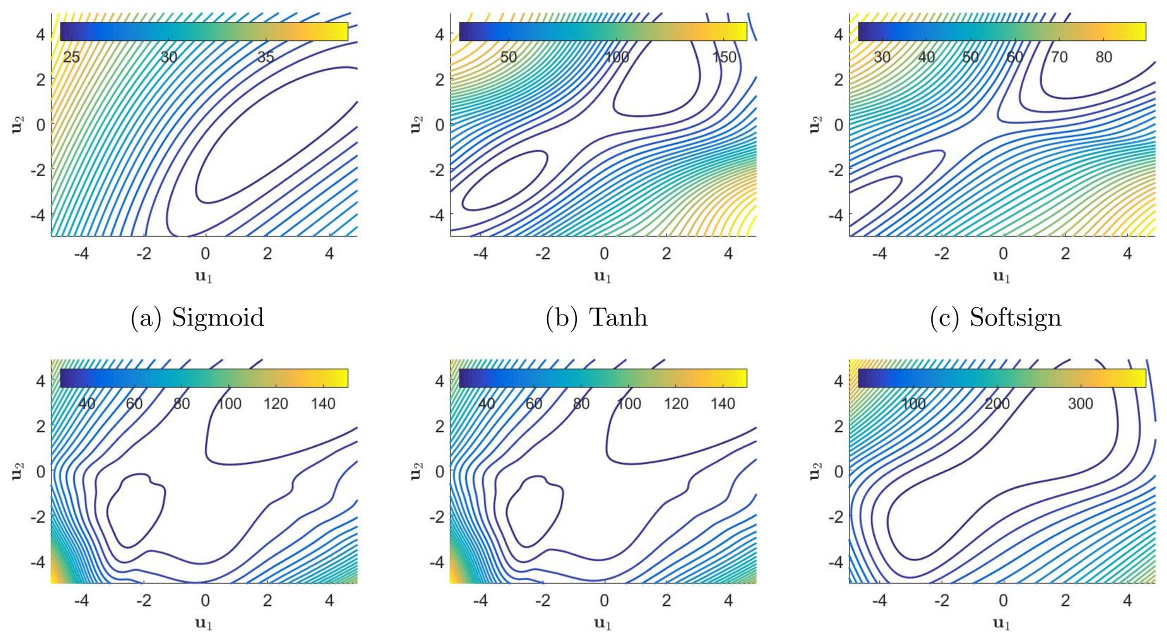
(c) Sparsity class function values (d) Sparsity class derivatives

Figure 6.1: (a,c) Function value and (b,d) derivatives of activation functions considered in our investigations. These are grouped together into (a,b) saturation and (c,d) sparsity classes respectively. The primary difference between saturation and sparsity classes are the derivatives in the positive input domain. The saturation class is characterized by derivatives that tend towards zero as input tends toward $+\infty$. Conversely, the sparsity class, is characterized by unit derivatives that spread over all of the positive input domain. This gives the sparsity class AFs behaviour characteristics that approximate a "switch", being either "on" or "off".

The respective function values and derivatives of both classes are shown in Figure 6.1 over an input domain of $[-5, 5]$. The saturation class is predominantly characterized by derivatives that tend to zero, as the inputs tend to $\pm\infty$. The function values begin either at 0 (as for Sigmoid) or -1 (Tanh and Softsign) have an upper limit of 1. Often, function values and derivatives of the saturation class are smooth and continuous. The outlier in the saturation AFs chosen for this survey is Softsign, which has a derivative that is continuous, but not smooth where the input is 0. The derivative characteristics of the Sigmoid AF are also notable among the saturation class AFs. The maximum derivative value of Sigmoid is 0.25, which is a factor of 4 lower than those of Tanh and Softsign with unit derivatives at the origin.

The original sparsity AF, ReLU, was introduced to recreate the sparseness and switching behaviour observed in neuroscientific studies, in artificial neural networks [Glorot and Bordes, 2011]. ReLU is characterized by having an output of zero in the negative input domain, and a linear output with unit gradient in the positive domain. This makes the function values of ReLU continuous and non-smooth, while the derivative is step-wise discontinuous at input 0. As with ReLU, the sparsity class is characterized by having linear outputs in the positive input domain, while the derivatives approximate zero as the negative input domains tend to $-\infty$. However, the derivatives in the positive input domains are always 1, which is a critical difference to the saturation class of AFs. The leaky ReLU AF relaxes the absolute sparsity of ReLU by allowing a small constant derivative in the negative input domain. However, the non-smooth function value and the discontinuous derivative properties of ReLU are maintained. The ELU AF is a further modification that enforces smoothness in the function value and continuity in the derivative. However, the derivative remains non-smooth. The formulations of leaky ReLU and ELU are both claimed to improve training performance over ReLU [Clevert et al., 2016].

We consider the difference in loss function characteristics of the selected AFs for a simple neural network problem presented in Figure 6.2. The contours of the mean squared error (MSE) loss function are depicted for a single hidden layer feedforward neural network with 10 hidden nodes, fitted to the Iris dataset [Fisher, 1936]. The plots are generated by taking steps $\alpha_{\{1\}} \in [-5, -4.5, \ldots, 5]$ and $\alpha_{\{2\}} \in [-5, -4.5, \ldots, 5]$ along the two random perpendicular unit directions, $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$, as inspired by Li et al. [2017].



(a) Sigmoid (b) Tanh (c) Softsign

(d) ReLU (e) leaky ReLU (f) ELU

Figure 6.2: Contours of the mean squared error loss function along two random perpendicular unit directions, $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$, computed for a single hidden layer feedforward neural network using different activation functions. The hidden layer consists of 10 hidden nodes and the architecture is applied to the classic Iris dataset [Fisher, 1936].

The contours of the Sigmoid AF represent a smooth loss function, containing a single minimum, where difference in function value is ± 15 over the sampled domain. The use of the Tanh

AF results in a larger range in function value of over ±120 and the emergence of an additional local minimum. Although the loss function range of Softsign is between that of Sigmoid and Tanh at ±60, in this case the two local minima drift further apart in the sampled domain. A characteristic feature of the saturation class of AFs is, that the loss function contours are smooth. Amongst the sparsity class of AFs, the ReLU AF retains the multi-modal nature of Tanh and Softsign, but demonstrates abrupt changes in contour characteristics. The modification of ReLU to include leaky gradients (leaky ReLU) softens these abrupt characteristics slightly, evidenced by the contours around the local minimum at $\alpha_{\{1\}} \approx -2$ and $\alpha_{\{2\}} \approx -2$. However, ELU impacts loss characteristics the most within the sparsity class, as it smooths out the contours of the loss and brings the two local minima closer together. ELU also results in a larger range of function value over the sampled domain, encompassing a change of ±300 compared to ±120 for ReLU and leaky ReLU respectively.

It is clear, that the choice of AF can significantly influence loss function landscape characteristics. By extension, these changes translate to the respective loss function gradients. Previous studies have confirmed, that loss functions with higher curvature cause SNN-GPPs to be more localized in space [Kafka and Wilke, 2019a]. Consequently, the aim is to investigate and quantify the choice of AF with regards to the performance of GOLS in determining step sizes for dynamic MBSS neural network training; and if it is adversely affected, what can be done to improve or restore performance.

6.2 Connections: Gradient-only line searches and activation functions

Consider neural network loss functions formulated as:

$$\mathcal{L}(\boldsymbol{x}) = \frac{1}{M}\sum_{b=1}^{M} \ell(\hat{\boldsymbol{t}}_b(\boldsymbol{x});\ \boldsymbol{t}_b), \tag{6.1}$$

where $\boldsymbol{x} \in \mathcal{R}^p$ denotes the vector of weights parameterizing the neural network, the training dataset of M samples is given by $\{\boldsymbol{t}_1, \ldots, \boldsymbol{t}_M\}$, and $\ell(\hat{\boldsymbol{t}}_b(\boldsymbol{x});\ \boldsymbol{t}_b)$ is the elemental loss evaluating $\boldsymbol{x}$ (via neural network model prediction $\hat{\boldsymbol{t}}_b(\boldsymbol{x})$) in terms of the training sample $\boldsymbol{t}_b$. Implementing backpropagation [Werbos, 1994] allows for efficient evaluation of the analytical gradient of $\mathcal{L}(\boldsymbol{x})$ with regards to $\boldsymbol{x}$:

$$\nabla\mathcal{L}(\boldsymbol{x}) = \frac{1}{M}\sum_{b=1}^{M} \nabla\ell(\hat{\boldsymbol{t}}_b(\boldsymbol{x});\ \boldsymbol{t}_b). \tag{6.2}$$

When $\mathcal{L}(\boldsymbol{x})$ and $\nabla\mathcal{L}(\boldsymbol{x})$ are evaluated using the full training dataset of M samples, the smoothness and continuity characteristics of both $\mathcal{L}(\boldsymbol{x})$ and $\nabla\mathcal{L}(\boldsymbol{x})$ are subject only to the smoothness and continuity characteristics of the AFs used in the neural network that constructs $\hat{\boldsymbol{t}}_b(\boldsymbol{x})$.

In order to conduct neural network training, the loss function in Equation (6.1) is minimized. Consider the standard gradient-descent update: $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \alpha\nabla\mathcal{L}(\boldsymbol{x}_n)$[Arora, 2011]. An iteration, n, is performed when the parameters, $\boldsymbol{x}_n$, are updated to a new state, $\boldsymbol{x}_{n+1}$. To determine step size, α, line searches are performed at every iteration, n, of the training algorithm in pursuit of a good minimum. Thus an iteration, n, encompasses exactly one line search. Line searches are conducted by finding the optimum of a univariate function, $F_n(\alpha)$, constructed from current solution, $\boldsymbol{x}_n$, along a search direction, $\boldsymbol{d}_n$. If full-batch sampling is implemented in univariate function $F_n(\alpha)$, we define:

$$\mathcal{F}_n(\alpha) = f(\boldsymbol{x}_n(\alpha)) = \mathcal{L}(\boldsymbol{x}_n + \alpha\boldsymbol{d}_n), \tag{6.3}$$

with corresponding directional derivative

$$\mathcal{F}'_n(\alpha) = \frac{dF_n(\alpha)}{d\alpha} = \boldsymbol{d}_n^T \cdot \nabla\mathcal{L}(\boldsymbol{x}_n + \alpha\boldsymbol{d}_n). \tag{6.4}$$

Figures 6.3(a) and (b) show examples of $\mathcal{F}_n(\alpha)$ and corresponding directional derivative $\mathcal{F}'_n(\alpha)$ for different AFs. For compactness, the explicit dependency on variables such as α is dropped in further discussions, unless specifically required. In Figure 6.3, $\mathcal{F}_n$ and $\mathcal{F}'_n$ are constructed along the normalized search direction $\boldsymbol{d}_n = \frac{\boldsymbol{u}_1+\boldsymbol{u}_2}{\|\boldsymbol{u}_1+\boldsymbol{u}_2\|_2}$ in our illustrative example introduced in Section 6.1. Note, how all instances of $\mathcal{F}_n$ are continuous. This means that minimization line searches [Arora, 2011] can be used to determine step sizes for training in full-batch sampled loss functions. The minimizer of $\mathcal{F}_n$, namely α^*, subsequently becomes the step size for iteration n, i.e. $\alpha_{n,I_n} = \alpha^*$, where I_n is the number of function evaluations required during the line search to find the optimum at iteration n. This notation can also be used to describe fixed step sizes. In such cases, α_{n,I_n} is a predetermined constant value over every iteration, and $I_n = 1$.

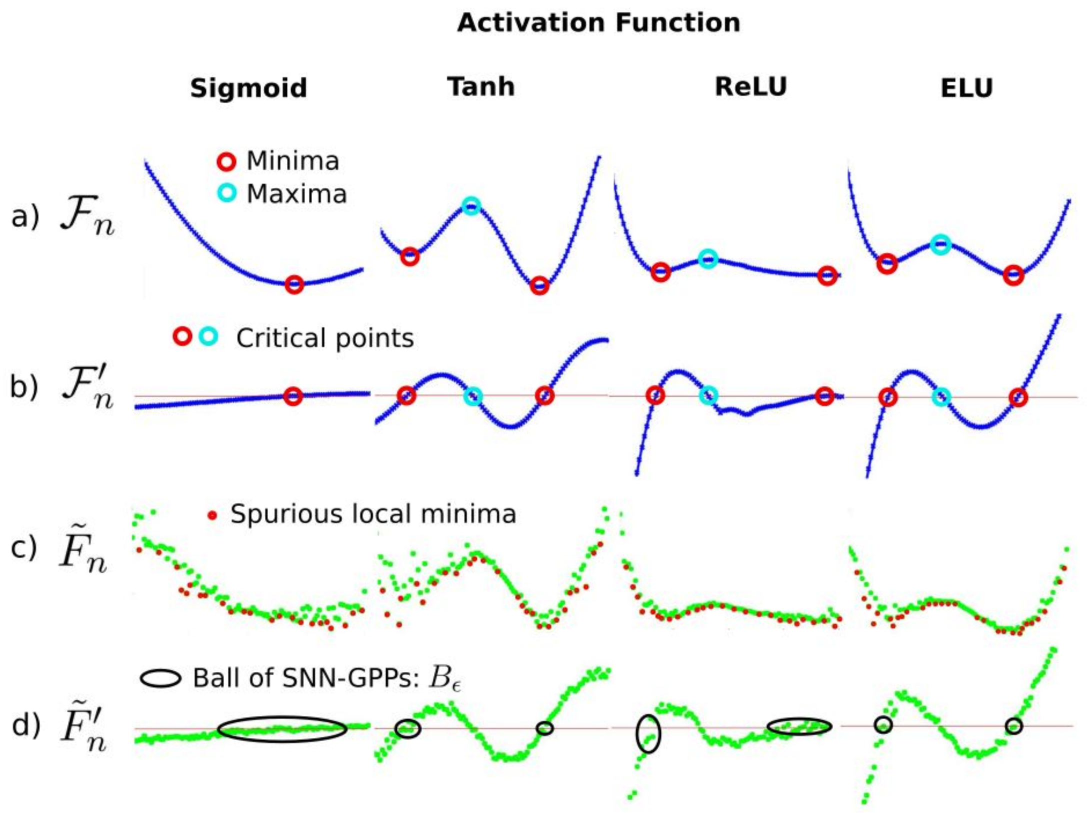


Figure 6.3: Univariate functions and directional derivatives of (a) $\mathcal{F}_n$ and (b) $\mathcal{F}'_n$, using full-batch sampling, and (c) $\tilde{F}_n$ and (d) $\tilde{F}'_n$, using dynamic mini-batch sub-sampling with a selection of activation functions.

However, modern datasets and corresponding network architectures have memory requirements that exceed current computational resources [Krizhevsky et al., 2012]. Therefore, only a fraction of the training data, $\mathcal{B} \subset \{1, \ldots, M\}$ with $|\mathcal{B}| \ll M$, is loaded into memory at a given time. This is referred to as mini-batch sub-sampling (MBSS). Omitting training data to construct mini-batches invariably results in a sampling error associated with the MBSS loss, compared to the full-batch sampled loss. Some training approaches employ static MBSS, where mini-batches are fixed for the minimum duration of a search direction, $\boldsymbol{d}_n$ [Friedlander and Schmidt, 2011, Bollapragada et al., 2017, Kungurtsev and Pevny, 2018, Kafka and Wilke, 2019b]. Alternatively, dynamic MBSS can be implemented, where a new mini-batch is resampled for every evaluation, i, of the loss function. It has been shown that dynamic MBSS, also referred to as approximate optimization [Bottou, 2010], can benefit a given training algorithm by exposing it to larger amounts of information per search direction [Kafka and Wilke, 2019b]. We therefore define dynamic MBSS approximations of $\mathcal{L}(\boldsymbol{x})$ and $\nabla\mathcal{L}(\boldsymbol{x})$ respectively, as:

$$\tilde{L}(\boldsymbol{x}) = \frac{1}{|\mathcal{B}_{n,i}|} \sum_{b \in \mathcal{B}_{n,i}} \ell(\hat{\boldsymbol{t}}_b(\boldsymbol{x});\ \boldsymbol{t}_b), \tag{6.5}$$

and

$$\tilde{\boldsymbol{g}}(\boldsymbol{x}) = \frac{1}{|\mathcal{B}_{n,i}|} \sum_{b \in \mathcal{B}_{n,i}} \nabla\ell(\hat{\boldsymbol{t}}_b(\boldsymbol{x});\ \boldsymbol{t}_b). \tag{6.6}$$

Note, that the mini-batch, $\mathcal{B}_{n,i}$, sampled for an instance of $\tilde{L}(\boldsymbol{x})$ and $\tilde{\boldsymbol{g}}(\boldsymbol{x})$ is consistent between the pair, while only at a new evaluation of pair $\tilde{L}(\boldsymbol{x})$ and $\tilde{\boldsymbol{g}}(\boldsymbol{x})$ is $\mathcal{B}_{n,i}$ resampled [Kafka and Wilke, 2019b]. However, the act of abruptly alternating between sampling errors associated different mini-batches $\mathcal{B}_{n,i}$, interrupts the smoothness and continuity characteristics of the loss function, irrespective of the choice of AF used in the neural network architecture. This results in point-wise discontinuous loss, $\tilde{L}(\boldsymbol{x})$, and gradient, $\tilde{\boldsymbol{g}}(\boldsymbol{x})$, functions. Although $\mathbb{E}[\tilde{L}(\boldsymbol{x})] = \mathcal{L}(\boldsymbol{x})$ and $\mathbb{E}[\tilde{\boldsymbol{g}}(\boldsymbol{x})] = \nabla\mathcal{L}(\boldsymbol{x})$ [Tong and Liu, 2005], the probability of encountering critical points, $\tilde{g}(\boldsymbol{x}^*) = \bar{\boldsymbol{0}}$, is infeasibly low in dynamic MBSS loss functions. Additionally, the point-wise discontinuities between consecutive evaluations of $\tilde{L}(\boldsymbol{x})$ result in the emergence of spurious candidate local minima [Wilson and Martinez, 2003, Schraudolph and Graepel, 2003, Schraudolph et al., 2007], which have been shown to be approximately uniformly distributed over the loss landscape [Kafka and Wilke, 2019a].

By substituting $\tilde{L}(\boldsymbol{x})$ and $\tilde{\boldsymbol{g}}(\boldsymbol{x})$ into Equations (6.3) and (6.4) respectively, we obtain dynamic MBSS univariate function $\tilde{F}_n(\alpha)$ and corresponding directional derivative $\tilde{F}'_n(\alpha)$. Consider Figures 6.3(c) and (d) for a range of AFs. Note, how sampling $\mathcal{B}_{n,i}$ uniformly from the full training dataset in Figure 6.3(c), results in local minima all along the search direction. Although directional derivatives Figure 6.3(d) get close to zero, none are a critical point, i.e. $\tilde{F}'_n(\alpha^*) \neq 0$. This is best illustrated by the first local minimum along the search direction for ReLU. Using full-batch sampling, a clear local minimum, $\mathcal{F}_n(\alpha^*)$, can be observed in Figure 6.3(a)(ReLU) with corresponding critical point, $\mathcal{F}'_n(\alpha^*) = 0$, in Figure 6.3(b)(ReLU). Both are indicated by the first red circle from left to right along $\mathcal{F}_n$ and $\mathcal{F}'_n$ respectively. Using dynamic MBSS in Figures 6.3(c)(ReLU) and (d)(ReLU), none of the directional derivatives are critical points, $\tilde{F}'_n \neq 0$, and local minima are located all along $\tilde{F}_n$, illustrated by small red points.

The discontinuities in $\tilde{F}_n$ make minimization ineffective in determining step sizes in dynamic MBSS loss functions [Kafka and Wilke, 2019b]. Historically, this has led to the popularity of subgradient methods for neural network training, using *a priori* determined step sizes.[Schraudolph, 1999, Boyd et al., 2003, Smith, 2015]. Line searches were first introduced into dynamic MBSS loss functions by Mahsereci and Hennig [2017], determining step sizes by using probabilistic surrogates along $\tilde{F}_n$ to estimate the location of optima. An alternative approach is the use of Gradient-Only Line Searches (GOLS) [Kafka and Wilke, 2019b, Kafka and Wilke, 2019], which employ an extension of the gradient-only optimality criterion [Wilke et al., 2013, Snyman and Wilke, 2018], namely the *Stochastic Non-Negative Associative Gradient Projection Point* (SNN-GPP) [Kafka and Wilke, 2019b], given as follows:

Definition 6.2.1. *SNN-GPP: A stochastic non-negative associated gradient projection point (SNN-GPP) is defined as any point, $\boldsymbol{x}_{snngpp}$, for which there exists $r_u > 0$ such that*

$$\nabla f(\boldsymbol{x}_{snngpp} + \lambda\boldsymbol{u})\boldsymbol{u} \geq 0,\ \ \forall\ \boldsymbol{u} \in \{\boldsymbol{y} \in \mathbb{R}^p \mid \|\boldsymbol{y}\|_2 = 1\},\ \ \forall\ \lambda \in (0, r_u], \tag{6.7}$$

with non-zero probability. [Kafka and Wilke, 2019b]

Subsequently, a ball, B_ϵ, exists that bounds all possible SNN-GPPs of a surrounding neighbourhood, where each neighbourhood contains one true optimum:

Definition 6.2.2. *B_ϵ: Consider a dynamic mini-batch sub-sampled loss function $\tilde{L}(\boldsymbol{x})$, of a continuous, smooth and convex full-batch loss function $\mathcal{L}(\boldsymbol{x})$, such that each sampled mini-batch with associated $L(\boldsymbol{x})$ that is used to evaluate $\tilde{L}(\boldsymbol{x})$ has the same smoothness, continuity and convexity characteristics as $\mathcal{L}(\boldsymbol{x})$. Then there exists a ball,*

$$B_\epsilon(\boldsymbol{x}) = \{\boldsymbol{x} \in \mathbb{R}^p : \|\boldsymbol{x} - \boldsymbol{x}^*\|_2 < \epsilon,\ 0 < \epsilon < \infty,\ \boldsymbol{x} \neq \boldsymbol{x}^*\}, \tag{6.8}$$

that contains all the stochastic non-negative gradient projection points (SNN-GPPs), where $\boldsymbol{x}^*$ is the minimum of $\mathcal{L}(\boldsymbol{x})$. [Kafka and Wilke, 2019b]

Along any univariate function, F_n, an SNN-GPP manifests as a sign change in the directional derivative from negative, $F'_n < 0$, to positive, $F'_n > 0$, along the descent direction. In the deterministic, full-batch setting, $\mathcal{F}_n$, the SNN-GPP reduces to the critical point associated with a local minimum, i.e. $\mathcal{F}_n(\alpha_{snngpp}) = \mathcal{F}_n(\alpha^*)$, and ball B_ϵ reduces to a single point. In the stochastic setting of dynamic MBSS losses, ball B_ϵ has a finite range that is dependent on the variance of the directional derivative as well as the expected curvature $\frac{\delta E[\tilde{F}'_n]}{\delta \alpha}$ in a neighbourhood [Kafka and Wilke, 2019a].

The SNN-GPP and B_ϵ can be visually illustrated in Figure 6.3(d). With the Sigmoid activation function, there is a single neighbourhood, with a large ball B_ϵ containing all SNN-GPPs. There exist numerous sign changes from negative to positive along the descent direction in B_ϵ, due to the variance of $\tilde{F}'_n$ and the slow change in $\tilde{F}'_n$ along α. In the case of Tanh, ReLU and ELU, there are two neighbourhoods in which SNN-GPPs can be found. These neighbourhoods are separated by a maximum, as demonstrated by $\mathcal{F}'_n$. Note, how the SNN-GPP definition ignores maxima, as it considers only sign changes from negative to positive along the descent direction. The $\tilde{F}'_n$ plots of Tanh, ReLU and ELU demonstrate how the size of ball B_ϵ decreases, with a decrease in variance and an increase in expected curvature. These $\tilde{F}'_n$ plots also show, that the characteristics of B_ϵ can change in different neighbourhoods of the loss function, and vary according to each AF.

It has been shown, that an exact GOLS will converge to an SNN-GPP within ball B_ϵ [Kafka and Wilke, 2019b]. Therefore, GOLS determine the step size at iteration n of a training algorithm, by locating an SNN-GPP such that $\alpha_{n,I_n} = \alpha_{snngpp}$. It has also been demonstrated, that the Gradient-Only Line Search that is Inexact (GOLS-I) behaves in a manner consistent with Lyapunov's global stability theorem [Lyapunov, 1992, Kafka and Wilke, 2019]. The latter proof was developed in the context of loss functions that are positive, coercive and AFs that result in strict descent [Kafka and Wilke, 2019]. Subsequently, this chapter explores how GOLS perform with a larger range of AFs; and consider the implications a given AF may have on a neural network architecture for a given problem.

6.3 Contribution

In this chapter we empirically study the interaction between activation functions and neural network architectures, when using Gradient-Only Line Searches (GOLS) to determine step sizes for dynamic MBSS loss functions. In our investigations we consider six activation functions, as introduced in Section 6.1, in the context of 1) shallow and deep feedforward classification networks, and 2) architectural features such as batch normalization and skip connections. To this end, we use 13 datasets to construct a range of training problems, where we primarily use the Gradient-Only Line Search that is Inexact (GOLS-I) [Kafka and Wilke, 2019b] to determine step sizes for training. Depending on the nature of the investigation, the Gradient-Only Line Search with Bisection (GOLS-B) [Kafka and Wilke, 2019b] and fixed step sizes are sporadically used to be benchmarks against which the performance of GOLS-I can be compared. Overall, our investigations demonstrate that GOLS-I is effective in determining step sizes in a range of feedforward neural network architectures using different activation functions. However, we also give examples, where activation function selection can significantly impede training performance with GOLS-I. We show that these difficulties can be alleviated by modifying the network architecture of a given problem. Therefore, this chapter serves as a practical guide for neural network practitioners to improve the construction of network architectures that promote efficient training using GOLS.

6.4 Empirical Study

In our studies we consider three different types of training problems, namely:

1. Foundational problems: Using small classification datasets with various feedforward neural network architectures.

2. MNIST with NetII: A training problem used by Mahsereci and Hennig [2017] to explore early stochastic line searches.

3. CIFAR10 with ResNet18: A state of the art architecture including skip-connections [He et al., 2016] and batch normalization [Ioffe and Szegedy, 2015].

The foundational training problems are used to explore the influence of AFs on training performance in the context of 1) hidden layer height and depth, 2) GOLS-I and GOLS-B, as well as constant step sizes; and 3) full-batch versus dynamic MBSS training. The NetII problem is used to demonstrate the potential sensitivity of training problems to the choice of AF, and subsequently show the corrective effect of batch normalization on training. Skip connections are another architectural consideration of interest with different AFs in the context of GOLS. The relationship between AFs and neural networks with skip connections is explored using the ResNet18 architecture with the CIFAR10 dataset, as adapted from the implementation by Liu [2018].

The datasets used in our investigations, along with and their respective properties, are listed in Table 6.1. All datasets were scaled using the standard transform (Z-score). For the foundational problems (spanning datasets 1 to 11), the standard transform was applied for each individual input D, while for MNIST and CIFAR10 the standard transform was applied over each image channel. For MNIST, there is a single grey scale channel of 28x28 pixels (total of 784 inputs), while CIFAR10 has 3 colour channels of 32x32 pixels each (total of 3072 inputs). Since the small datasets are not separated into training and test datasets by default, we divided them manually into training, validation and test datasets with a ratio of 2:1:1 respectively. We choose this division to demonstrate that the manual construction of validation and test datasets resulted in representative, unbiased hold-out datasets. Therefore, we expect similar performance between validation and test datasets. Conversely, both MNIST and CIFAR10 datasets have been predetermined test datasets, which are subsequently used.

No.	**Dataset name**	**Author**	**Observations**	**Inputs**, D	**Classes**, K
1	Iris	Fisher [1936]	150	4	3
2	Glass1	Prechelt [1994]	214	9	6
3	Horse1	Prechelt [1994]	364	58	3
4	Forests	Johnson et al. [2012]	523	27	4
5	Simulation failures	Lucas et al. [2013]	540	20	2
6	Soybean1	Prechelt [1994]	683	35	19
7	Card1	Prechelt [1994]	690	51	2
8	Cancer1	Prechelt [1994]	699	9	2
9	Diabetes1	Prechelt [1994]	768	8	2
10	Heartc1	Prechelt [1994]	920	35	2
11	Biodegradable compounds	Mansouri et al. [2013]	1054	41	2
12	MNIST	Lecun et al. [1998]	70000	784	10
13	CIFAR10	Krizhevsky and Hinton [2009]	60000	3072	10

Table 6.1: Properties of the datasets considered for conducted investigations.

Table 6.2 summarizes the 11 investigations performed in this chapter on the corresponding neural network training problems. For the foundational problems we implement shallow nets with the number of hidden nodes being half of the input dimensions, $\frac{D}{2}$ and twice that of the input dimensions, $2D$. We also implement a deep architecture with 6 hidden layers of $2D$ nodes. We conduct training limited by iterations for all training problems except NetII, which is limited in the number of function evaluations, as prescribed by Mahsereci and Hennig [2017]. We couple each of the training problems with the activation functions discussed in Section 6.2. Constructed dynamic MBSS, search directions $\boldsymbol{d}_n = -\tilde{\boldsymbol{g}}(\boldsymbol{x})$ were supplied by the line search stochastic gradient descent (LS-SGD) algorithm [Robbins and Monro, 1951, Kafka and Wilke, 2019b] for all training runs accept those of the deep architecture applied to the foundational problems, which uses Adagrad [Duchi et al., 2011, Kafka and Wilke, 2019b] instead. We adopt the convention whereby the name of a method is the combination of the line search used to determine the step size, and the algorithm used to provide the search direction. For example, using GOLS-I to determine step sizes for Adagrad is denoted "GOLS-I Adagrad". As indicated in Table 6.2, step sizes for LS-SGD were predominantly determined using GOLS-I, while alternatively GOLS-B and fixed step sizes were implemented, depending on the investigation performed. The fixed step size, $\alpha_{n,I_n} = 0.05$, used in investigation 4 was manually tuned to give good training performance for the foundational problems with the ReLU AF. The fixed step sizes chosen for investigation 8 were selected in order to highlight a range of training performance, from slow to unstable, for the NetII training problem with the ReLU AF.

Inv. #	**Data-set #**	**Hidden layer structure**	**Batch norm.**	**Step size method**	**Training limit**	$\lvert\mathcal{B}_{n,i}\rvert$	**Loss**
1	1-11	$[\frac{D}{2}]$	No	GOLS-I SGD	3000 Its.	32	MSE
2	1-11	$[2D]$	No	GOLS-I SGD	3000 Its.	32	MSE
3	1-11	$[2D, 2D, 2D, 2D, 2D, 2D]$	No	GOLS-I Adagrad	3000 Its.	32	MSE
4	1-11	$[2D]$	No	$\alpha_{n,I_n} = 0.05$ with LS-SGD	3000 Its.	32	MSE
5	1-4	$[2D]$	No	GOLS-B SGD	3000 Its.	64	MSE
6	1-4	$[2D]$	No	GOLS-B SGD	3000 Its.	M	MSE
7	12	$[1000, 500, 250]$	No	GOLS-I SGD	40,000 FEs	100	MSE
8	12	$[1000, 500, 250]$	No	$\alpha_{n,I_n} = 0.1$, $\alpha_{n,I_n} = 0.01$, $\alpha_{n,I_n} = 0.001$ with LS-SGD	40,000 FEs	100	MSE
9	12	$[1000, 500, 250]$	Yes	GOLS-I SGD	40,000 FEs	100	MSE
10	13	ResNet18	Yes	GOLS-I SGD	40,000 Its.	128	BCE
11	13	ResNet18	No	GOLS-I SGD	40,000 Its.	128	BCE

Table 6.2: Parameters and settings governing the implemented network architectures and their training for various investigations (Inv.).

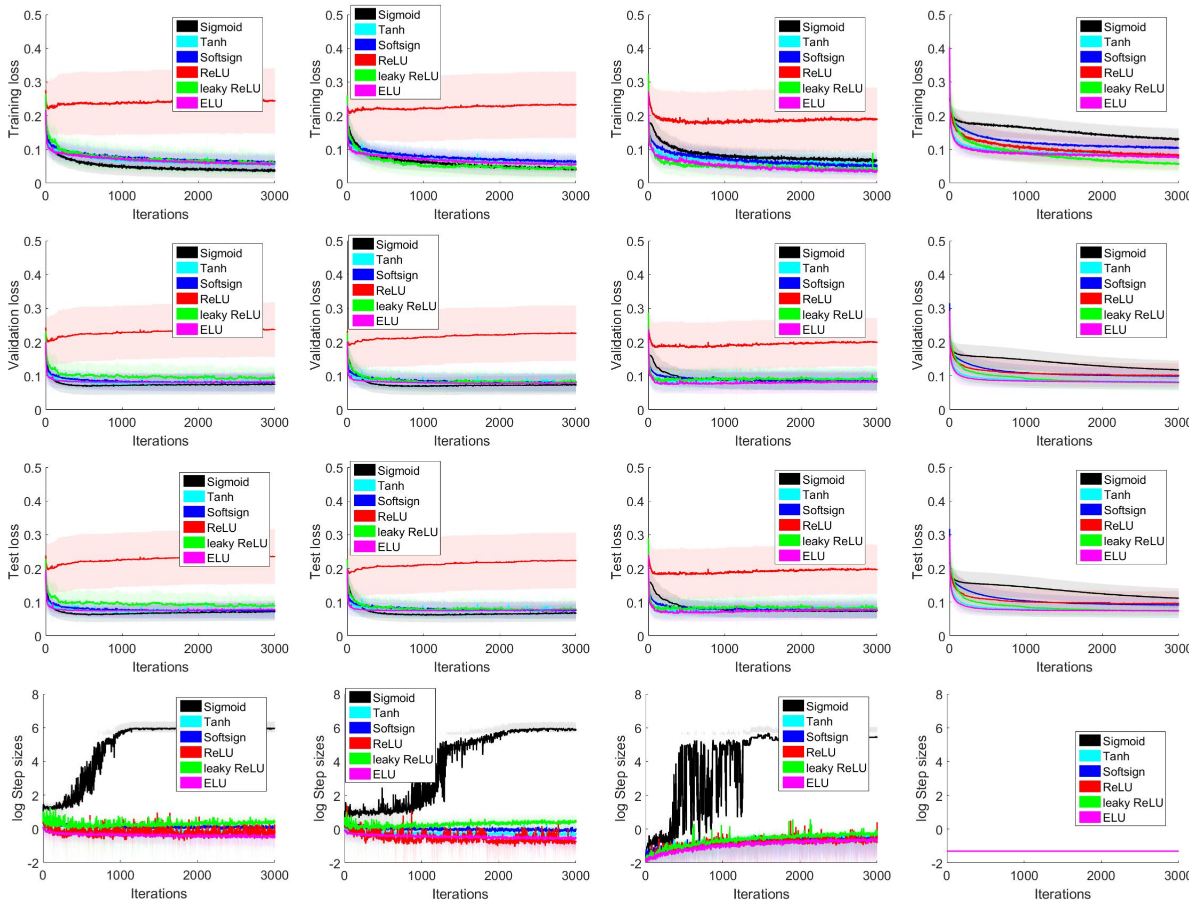
All training runs were conducted using PyTorch 1.0 [pytorch.org, 2019]. By default, He initialization [He et al., 2015] is used for networks implementing ReLU and leaky ReLU AFs, while Xavier initialization [Glorot and Bengio, 2010] is used for networks with the remaining AFs considered in this survey. For the foundational and NetII training problems, 10 training runs were conducted for each dataset and AF combination, whereas for CIFAR10 with ResNet18, a single training run per AF is performed.

6.5 Results

The results of our empirical study are ordered according to the training problems considered, namely 1) foundational problems, 2) the MNIST dataset with the NetII architecture, and 3) the CIFAR10 dataset with the ResNet18 architecture. Note, that the loss is used to evaluate training performance for the foundational problems, while the classification errors of the datasets are plotted to evaluate NetII and ResNet18. Note, that results given in terms of iterations are not representative of computational cost, as a number of function evaluations can be performed per iteration when line searches are implemented. However, giving results in terms of iteration allows for comparison between line searches with different costs, while assessing the reduction in loss (i.e. the quality) provided by the line searches.

6.5.1 Foundational problems

The average training, validation and test losses with corresponding average step sizes for the foundational problems are given in Figure 6.4. The results of the foundational problems are averaged over all the respective datasets and their corresponding 10 training runs. This allows a representative trend to be demonstrated for each AF over a number of datasets. The results of the first analysis, with hidden layers of size $\frac{D}{2}$, are shown in the first column, Figure 6.4(a). Firstly, it is evident that step sizes estimated by GOLS-I result in effective training over a range of datasets and AFs. The mean training loss continually decreases, while that of both the validation and test datasets increases after 500 iterations, indicating the onset of overfitting. The consistency between validation and test losses suggests, that both validation and test datasets are large enough to give unbiased assessment of the neural networks' generalization performance.



(a) $\frac{D}{2}$, GOLS-I SGD (b) $2D$, GOLS-I SGD (c) $2D(6)$, GOLS-I Ada. (d) $2D$, $\alpha_{n,I_n} = 0.05$

Figure 6.4: Training, validation and test losses with corresponding log of step sizes for the foundational problems with various networks architectures, including (a) investigation 1: single hidden layer (HL) networks with $\frac{D}{2}$ hidden nodes, (b) investigation 2: single HL networks with $2D$ nodes and (c) investigation 3: networks with six HLs of $2D$ nodes using GOLS-I Adagrad. For (a) to (c), GOLS-I was used to determine step sizes, while (d) investigation 4 implements LS-SGD with fixed step sizes $\alpha_{n,I_n} = 0.05$ for a single HL network with $2D$ nodes.

In investigation 1, shown in Figure 6.4(a), ReLU is convincingly the worst performer. Conversely, the Sigmoid AF is the best performer in training, while the performance between the remaining AFs is almost indistinguishable. The best validation and test losses also belong to Sigmoid, with a marginal advantage over the remaining AFs (accept ReLU, where the difference is significant). Interestingly, the step sizes of Sigmoid, presented in the last row of Figure 6.4(a), diverge significantly from those of the remaining AFs. We postulate that this phenomenon compensates for the Sigmoid's small derivative magnitudes, see Figure 6.1(b). As mentioned, the maximum derivative magnitude of the Sigmoid AF is 0.25, while for the remaining AFs it is 1. Cumulatively, this results in progressively smaller gradients for Sigmoid, as the training algorithm progresses closer towards an optimum. Subsequently, this prompts GOLS-I to increase the step sizes in the search for univariate SNN-GPPs. This is not common among the foundational problem datasets, but the larger step sizes of individual problems dominate the calculation of average step sizes.

The ReLU AF was introduced to promote sparsity within a neural network, which is meant to approximate brain processes observed in neuroscience [Glorot and Bordes, 2011], where only a fraction of the network is used at a given time. However, ReLU was developed predominantly for large neural networks, which makes it unclear whether the bad performance observed in Figure 6.4(a) is due to the use of GOLS-I for determining learning rates, or due to the network architecture selected being too small.

We therefore perform training runs with increased hidden layer sizes of $2D$ for investigation 2 shown in Figure 6.4(b). Hence, the hidden layers of network architectures trained in investigation 2 are 4 times larger than those considered in investigation 1. However, this increase has little effect on improving the training performance of ReLU. Instead, there is a shift between the relative performances of the remaining AFs. Subsequently, leaky ReLU outperforms the Sigmoid AF in training. However, this does not translate to generalization, where the Sigmoid still outperforms leaky ReLU.

Presuming that the investigated architectures are still too small to cater for the sparsity that ReLU induces, we increase the number of hidden layers to 6 with $2D$ nodes each in investigation 3. To aid training with deeper layers, we employ the GOLS-I Adagrad for this analysis. The results shown in Figure 6.4(c) exhibit an average loss improvement of 0.0457 for ReLU over the single hidden layer training runs performed for investigation 2 in Figure 6.4(b). It is notable, that the performance rankings between the remaining activation functions has again changed. In this case, the other AFs of the sparsity class (leaky ReLU and ELU), begin to dominate both training and generalization over AFs of the saturation class. The increasing step sizes in Figure 6.4(c)(last row) occur as GOLS-I corrects for the diminishing norm of Adagrad's search directions [Duchi et al., 2011], which is expected [Kafka and Wilke, 2019b]. Although GOLS-I determines useful learning rates that result in effective training for a range of AFs for a significantly larger network architecture, this analysis has failed to demonstrate a significant improvement in the training performance of ReLU with GOLS-I. This suggests, that neither architecture nor the search direction are dominant factors in explaining ReLU's performance in our experiments thus far. Therefore, the use of GOLS-I to determine the step sizes for ReLU architectures comes under closer scrutiny.

We determine GOLS-I's contribution to poor ReLU performance, by process of elimination. In investigation 4, shown in Figure 6.4(d), we substitute GOLS-I for the use of a manually tuned fixed step size of $\alpha_{n,I_n} = 0.05$ with LS-SGD for all AFs. It is clear, that the training performance with ReLU improves significantly with $\alpha_{n,I_n} = 0.05$. However, this is not coincidental, as the fixed step size was manually tuned for this purpose. This indicates, that the step sizes determined by GOLS-I are not effective for ReLU with the foundational training problems. Interestingly, the variance between training performances of the remaining AFs is higher with LS-SGD using fixed step size, than when implementing GOLS-I SGD. The Sigmoid AF performs significantly worse, due to its comparatively smaller derivatives, as discussed above. This confirms, that GOLS-I is capable of adapting its step sizes to the properties of different AFs in feedforward network architectures, with the exception of ReLU.

6.5.2 A closer look at ReLU loss functions using GOLS-B SGD

What makes ReLU the outlier among the considered AFs, is that it enforces sparsity in an absolute manner, i.e. the derivative in the negative input domain is exactly zero. Previous studies have shown, that the use of ReLU with the MSE loss can cause $\tilde{L}(\boldsymbol{x})$ and $\tilde{\boldsymbol{g}}(\boldsymbol{x})$ to be zero over a range of $\boldsymbol{x}$ [Kafka and Wilke, 2019a]. This breaks the assumptions of positivity, coerciveness and strict descent, namely those of Lyapunov's global stability theorem, which govern the convergence of GOLS [Kafka and Wilke, 2019]. The reason for GOLS-I's inability to train feedforward networks with ReLU is as follows: Conducting updates with step sizes that are too large, as is possible in an inexact line search such as GOLS-I, can cause numerous nodes within a ReLU network to enter the negative input domain. If the step sizes are large enough, the affected nodes remain "off" irrespective of the variance in the incoming data. Subsequently, large parts of the network may be "off" permanently, causing the gradient vector to have numerous zero-valued partial derivatives, i.e. become sparse. This results in no change to weights with zero-partial-derivatives during update steps, effectively terminating training for the deactivated portion of the architecture.

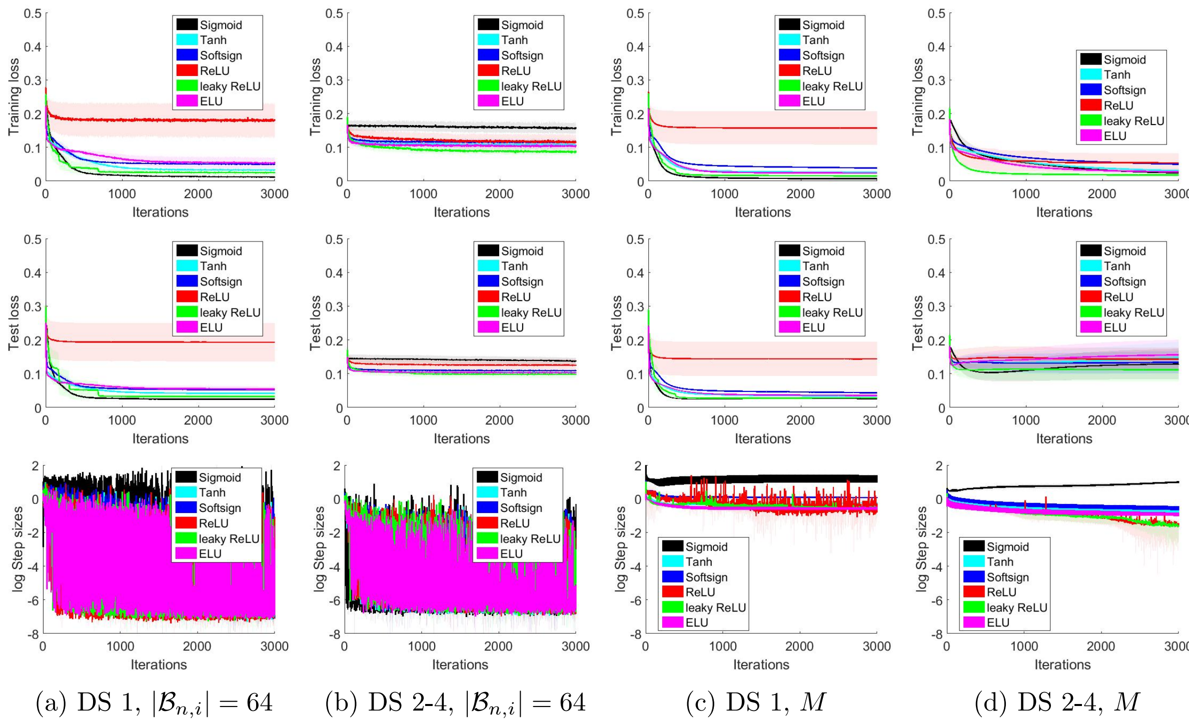


Figure 6.5: Training loss, test loss and the log of step sizes for a subset of datasets (DS) from the foundational problems dataset pool, trained using GOLS-B SGD (a,b) with mini-batch size $|\mathcal{B}_{n,i}| = 64$ for investigation 5 and (c,d) using full-batches, M in investigation 6.

These considerations, as well as the results observed in Figure 6.4, suggest that the step sizes determined by GOLS-I are too large to result in stable training with ReLU. A contributing factor is that GOLS-I's initial accept condition [Kafka and Wilke, 2019b] allows univariate SNN-GPPs to be overshot in a controlled manner. Although overshoot has been shown to aid the training performance of LS-SGD [Kafka and Wilke, 2019b, Kafka and Wilke, 2019], it may be too aggressive to be implemented with ReLU AFs. Therefore, investigations 5 and 6 focus on whether GOLS-B [Kafka and Wilke, 2019b] with a conservative SNN-GPP bracketing strategy is capable of determining step sizes for LS-SGD in feedforward networks with ReLU AFs. The conservative bracketing strategy grows the minimum step size by a factor of 2, until a positive directional derivative is observed. This increases the probability of encountering SNN-GPPs in B_ϵ that are closest to $\boldsymbol{x}_n$ along the descent direction.

In investigations 5 and 6, we more closely consider the loss functions of the first 4 foundational problems with ReLU AFs. The focus is primarily on the change in characteristics between dynamic MBSS and full-batch sampled losses. Therefore, we find the largest mini-batch size of the power 2 that allows MBSS for the first 4 problems. Due to separating problem data into training, validation and test datasets, the training dataset sizes for the first 4 problems are $M \in \{76, 108, 182, 263\}$ respectively. Therefore, the largest common mini-batch size of power 2 is $|\mathcal{B}| = 64$, which is implemented for investigation 5. Subsequently, investigation 6 uses full-batch sampling for the same datasets. The results for both investigations shown in Figure 6.5, where all step sizes are determined using GOLS-B. Satisfied, that the validation and test datasets have been chosen representatively in Figure 6.4, we omit the validation dataset losses in Figure 6.5. We also separate the results of dataset 1 in Figures 6.5(a) and (c) from those of datasets 2-4 in Figures 6.5(b) and (d), as these have distinct training characteristics.

The results for training on dataset 1, which is the smallest considered dataset among the foundational problems, with $|\mathcal{B}_{n,i}| = 64$ on the single hidden layer architecture with $2D$ nodes are shown in Figure 6.5(a). Here, training the ReLU architecture is still unstable, while training the networks of the other AFs is effective. Note, that the step sizes of all AFs are significantly smaller and noisier with GOLS-B SGD compared to those of GOLS-I SGD. This is due to a combination of the conservative bracketing strategy and the lack of overshoot, compared to GOLS-I. However, this conservative approach aids in successfully training feedforward networks with the ReLU AF for datasets 2-4 in Figure 6.5(b). This is confirmation, that the larger step sizes determined by GOLS-I led to the divergent training behaviour observed in Figure 6.4. However, this improved stability comes at the expense of training performance, as GOLS-B is an exact line search, which uses an order of magnitude more function evaluations per iteration compared to GOLS-I [Kafka and Wilke, 2019b].

The full-batch sampled loss or true loss function results for investigation 6 are plotted in Figures 6.5(c) and (d). The training performance of dataset 1 with ReLU in Figure 6.5(c) shows little significant improvement in comparison to Figure 6.5(a). The average training loss is only marginally better, with a mean drop in loss of 0.022 for full-batch training. This indicates, that the deterministic loss function pertaining to dataset 1 with ReLU has descent directions leading into flat planes, which trap the training algorithm.

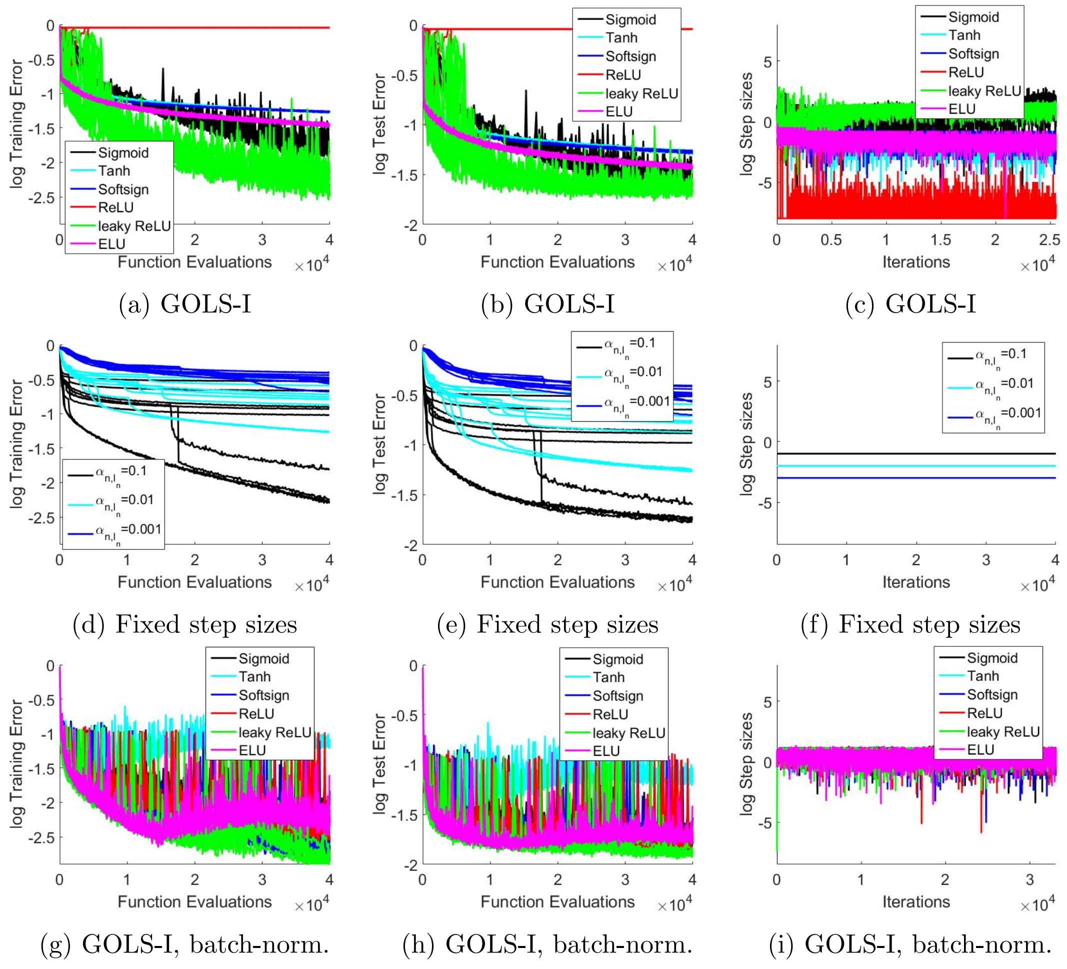
Generally, the determined step sizes for all AFs are significantly more stable for the full-batch case, than with $|\mathcal{B}_{n,i}|$. There are no incidences of minimum step sizes, as all search directions are deterministic descent directions. The variance that remains is due to the qualities of the deterministic descent direction itself, where the step size to the SNN-GPP along each descent direction is different. However, the step sizes of ReLU networks are particularly noisy, as GOLS-B SGD contends with the boundary between obtaining gradients that are dense, or sparse. Compared to dataset 1, this variance in step size is significantly reduced for datasets 2-4, which indicates that the boundary between obtaining dense and sparse gradient vectors along a descent direction is less abrupt, prompting less aggressive changes in step sizes. This is echoed by the training performance for ReLU with datasets 2-4, which is competitive with that of the remaining AFs. This suggests, that some ReLU feedforward network architectures construct small positive directional derivatives that "push" the line search back from zero-valued domains in the loss function, an observation confirmed by Kafka and Wilke [2019a].

An additional interesting observation between Figures 6.5(b) and 6.5(d) is that the minimum test losses of most AFs are lower in the dynamic MBSS case, than when using full-batch training. It is clear, that training stagnates in Figure 6.5(b) compared to Figure 6.5(d), where the training loss decreases at a more rapid rate. However, as training slows in Figure 6.5(b), the test losses for all AFs accept for Sigmoid are lower than their full-batch equivalent in Figure 6.5(d). This indicates, that dynamic MBSS together with a conservative gradient-only line search can either slow training (as is the case for Sigmoid), or act as a regularizer during training (as is the case for all other AFs), which definitely warrants future study.

This investigation has demonstrated, that successfully determining step sizes using GOLS for feedforward neural networks with ReLU AFs is sensitive to both the line search strategy used, and the architecture of the given problem. In the examples shown, GOLS-B effectively resolved step sizes for LS-SGD in larger networks with ReLU AFs, while GOLS-I SGD was unable to conduct reliable training on the same problems. Step sizes that are too large, and variance produced by dynamic MBSS, lead to detailed features such as small positive directional derivatives being ignored. This impedes the ability of GOLS-I to determine effective step sizes for the ReLU AF. GOLS-B resolves more conservative step sizes, but bears a high computational cost. Instead, relaxing the absolute sparsity of ReLU, by implementing the leaky ReLU or ELU AF, significantly improves GOLS-I's ability to determine effective step sizes in dynamic MBSS losses for the feedforward architectures considered in this survey. The non-zero derivatives of ELU and leaky ReLU in their negative input domains ensure that $\tilde{\boldsymbol{g}}(\boldsymbol{x})$ remains dense. This guarantees that all weights participate in update steps, and allows the training algorithm to recover from previous step sizes that were too large.

6.5.3 MNIST with NetII

Next, we present a larger training problem, where the choice of AF significantly influences training performance using GOLS-I SGD. The NetII architecture in combination with the well known MNIST dataset has been used to demonstrate the ability of probabilistic line searches to determine learning rates in dynamic MBSS loss functions [Mahsereci and Hennig, 2017]. Mahsereci and Hennig [2017] only implement the Tanh AF for this problem, which we extend by analysing all the AFs considered in Section 6.2 for investigation 7. In addition, we quantify the effect of batch normalization in investigation 9. The training and test classification errors, as well as accompanying step sizes for the different AFs are given in Figure 6.6. We remind the reader, that the results presented for investigations 7-8 are given in function evaluations to be consistent with Mahsereci and Hennig [2017].



(a) GOLS-I (b) GOLS-I (c) GOLS-I

(d) Fixed step sizes (e) Fixed step sizes (f) Fixed step sizes

(g) GOLS-I, batch-norm. (h) GOLS-I, batch-norm. (i) GOLS-I, batch-norm.

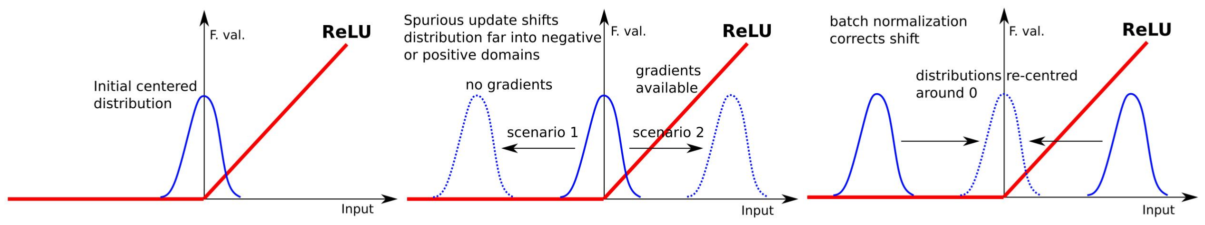
Figure 6.6: Training and test classification errors with corresponding log of step sizes as obtained for the MNIST Dataset with the NetII architecture for different activation functions. Step sizes for LS-SGD are determined by (a)-(c) GOLS-I for the standard NetII architecture with all considered AFs, (d)-(f) a range of fixed step sizes for the standard NetII architecture with only ReLU, and (g)-(i) determined by GOLS-I for NetII with batch normalization and all considered AFs.

In investigation 7, the training performance between different AFs varies significantly for the standard NetII architecture with GOLS-I SGD in Figures 6.6(a)-(c). It is immediately evident, that ReLU is also unstable in the standard NetII architecture using GOLS-I SGD. The overall best performance is obtained using leaky ReLU, followed by Sigmoid. The next best performer is ELU, with Softsign marginally outperforming Tanh. Leaky ReLU trains particularly noisily in the first 5,000 function evaluations, before establishing a clear lead over the remaining AFs. Sigmoid trains slower than Tanh, Softsign and ELU during the first 10,000 function evaluations, but outperforms these thereafter. The relative performances of the AFs generalize, as they are also reflected in the test classification errors. There are a few indications that the resulting loss function of leaky ReLU and the Sigmoid AFs have different characteristics to those of Tanh, Softsign and ELU, namely: 1) The significantly lower training and test errors after 40,000 function evaluations, 2) the higher variance in error during training, and 3) step sizes that are up to two orders of magnitude larger than those of the rest. We speculate that this is due to specific interactions between activation function properties and the neural network architecture.

Both Sigmoid and leaky ReLU are AFs that "activate" in the positive input domain and tend towards zero in the negative input domain. Therefore, we postulate that the ability of the AFs to approximate zero function value outputs, while having non-zero derivatives, constructs loss function landscapes that are easier to traverse with GOLS-I SGD. This supports a study by Xu et al. [2016], which found that a "penalized Tanh", that reduces the output magnitudes of function values and derivatives of Tanh in the negative input domain, performed competitively with sparsity class activation functions in deep convolutional neural network training. We postulate that the ability to significantly reduce the absolute function value of a node, decreases the information passed forward into a network. Subsequently, this reduces the sensitivity of nodes further downstream to the nodes that have low function value output. This contributes towards uncoupling parameters in the optimization space, $\boldsymbol{x}$, thus changing the nature of the loss function and resulting the unique training characteristics observed for Sigmoid and leaky ReLU.

Since ReLU demonstrates the same training difficulties with GOLS-I SGD as investigated in Section 6.5.1, we consider training ReLU using LS-SGD with fixed step sizes of $\alpha_{n,I_n} \in \{0.1, 0.01, 0.001\}$ for investigation 8 in Figures 6.6(d)-(f). Again, training performance improves for ReLU using LS-SGD with fixed step sizes over GOLS-I SGD. However, the variance between each of the 10 training runs performed increases proportionately to the fixed step size. Compared to the relatively consistent performance of the other AFs with GOLS-I SGD, this result is unsatisfactory. Although individual training runs with $\alpha_{n,I_n} = 0.1$ outperform GOLS-I with Sigmoid or leaky ReLU, the fixed step size first needs to be determined, and subsequently multiple runs performed to obtain an appropriate α_{n,I_n}. This highlights, that training using fixed step sizes is not a practical alternative to using GOLS-I with LS-SGD in feedforward neural networks and ReLU activation functions. As argued in Section 6.5.1 implementing GOLS-B instead is computationally too demanding. It is therefore preferable to explore alternative means, by which the benefits of GOLS-I can be extended to ReLU architectures.

The problem of training ReLU feedforward architectures using GOLS-I, as considered in Section 6.5.1, is summarized in Figure 6.7. At initialization, the distribution of information entering ReLU's input domain is centred around 0 [He et al., 2016], see Figure 6.7(a). This allows the "switching" mechanism proposed for ReLU to occur, whereby some data samples cause the node to "fire" (inputs in the positive domain) and others keep the node dormant (inputs in the negative domain). As discussed, large variance in gradient norms and the nature of a line search can cause step sizes that are spuriously too large, resulting in large changes in weight updates. Such updates can shift all the training dataset variance propagated through the network far into the negative or positive input domain of ReLU, see Figure 6.7(b). If all the data variance is in the positive input domain (scenario 2), gradients are available to allow subsequent update steps to correct for this shift. However, if all the dataset variance is in the negative input domain (scenario 1), the zero-derivative of ReLU prohibits information from travelling through the activation function to subsequent nodes. When this occurs to a significant portion of the network architecture, the flow of information through the network is significantly hampered and training stagnates.


F. val.
ReLU
Initial centered distribution
Input
Spurious update shifts distribution far into negative or positive domains
F. val.
ReLU
no gradients
gradients available
scenario 1
scenario 2
Input
batch normalization corrects shift
F. val.
ReLU
distributions re-centred around 0
Input

(a) At initialization (b) Spurious update w/o BN (c) With batch normalization

Figure 6.7: Schematic of the distribution of incoming information to a ReLU activation function (a) at initialization, (b) without batch normalization (BN) after spurious updates due to step sizes that are too large, and (c) after implementing batch normalization. Batch normalization centres the incoming information distribution around 0 and scales it, ensuring that the ReLU activation function remains active with a reasonable probability. Subsequently, $\tilde{\boldsymbol{g}}(\boldsymbol{x})$ is more likely to remain dense than with the standard feedforward architecture.

Batch normalization [Ioffe and Szegedy, 2015] is a method by which the inputs to a layer of nodes are continually centred by the mean of the previous layer's output and scaled by the corresponding variance. Applying batch normalization results in the distribution of information into a node remaining around the centre of the ReLU input domain, see Figure 6.7, increasing the likelihood of $\tilde{\boldsymbol{g}}(\boldsymbol{x})$ for the whole architecture remaining dense. This should increase the ability of GOLS-I to determine step sizes for ReLU, as partial derivatives are more likely to be non-zero, even after spurious updates.

We implement batch normalization for NetII with the given AFs for investigation 9 and show the results in Figures 6.6(g)-(i). It is clear, that the training performance of ReLU is drastically improved with GOLS-I. For the first time, it is possible to obtain competitive performance with ReLU using GOLS-I. Additionally, all activation functions show accelerated training with batch normalization over the standard NetII architecture. The best training performances are shared by Leaky ReLU and Softsign. However, the test errors are more comparable, for all AFs, ranging between $10^{-1.5}$ and 10^{-2} after 40,000 function evaluations. One exception is Tanh, which diverges after 5,000 function evaluations. However, the training and test errors achieved with batch normalization before 5,000 function evaluations are lower than those achieved for the standard architecture after 40,000 function evaluations. Therefore, although the reason for Tanh's divergence is worthy of further investigation, in this study we are satisfied with observing improved performance with Tanh due to batch normalization.

It is noteworthy, that the step sizes for NetII with batch normalization have a consistent magnitude between different AFs and vary less in comparison to those determined for the standard NetII architecture. Since batch normalization alters the scaling of the search direction, the loss function seems more spherical to LS-SGD, which results in the step sizes being more consistent between AFs. However, this does not result in equal training performance between AFs (as seen for Tanh), which indicates that the different AFs still contribute unique characteristics to the loss function. Interestingly, the error variance characteristics also change, for many AFs. Leaky ReLU and Sigmoid errors remain noisy, but AFs such as Tanh, Softsign and ELU exhibit more variance with batch normalization than without, as their respective derivatives are highest around 0.

In summary, this investigation has shown that:

1. Larger feedforward networks (than investigated in Section 6.5.1) with ReLU AFs can also be unstable when training with GOLS-I.

2. The interaction between neural network architecture and AF can lead to significant differences in training performance with GOLS-I (even when the systemically poor performance of ReLU is ignored).

3. Implementing constant learning rates is not a viable alternative to determining step sizes

with GOLS-I for ReLU AFs.

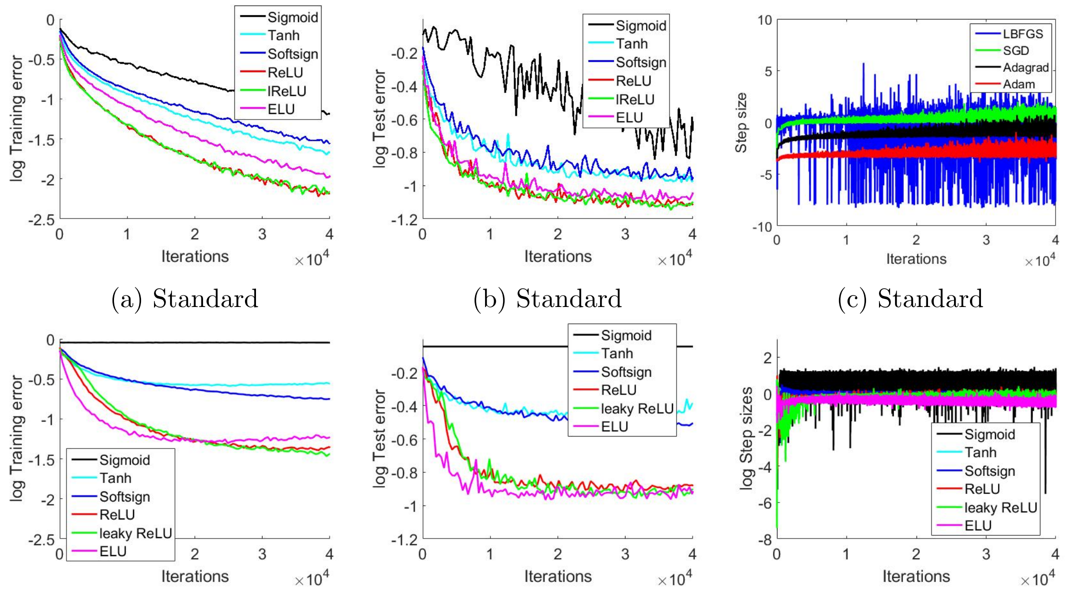
4. Instead, batch normalization significantly improves training of ReLU architectures using GOLS-I.

5. And lastly, batch normalization accelerated training for all AFs with GOLS-I in our investigation.

6.5.4 CIFAR10 with ResNet18

Consider the interaction between architectures that use skip connections [He et al., 2016] and different AFs using GOLS-I SGD. To this end, we implement the ResNet18 architecture, as applied to the CIFAR10 dataset. Skip connections ensure that the information flow to subsequent nodes remains unimpeded, regardless of whether an activation function such as ReLU prohibits information from travelling through a given node. The role of AFs in this case is to manipulate information that is additional to that transferred by the skip connections i.e. the "residuals". The interaction between the "skip-transferred" information and the residuals constructs the mapping behaviour between input and output domains of the neural network. The standard ResNet18 architecture includes batch normalization.

For investigation 10, we compare training and test classification error, as well as corresponding step sizes for ResNet18 with the considered range of AFs in Figures 6.8(a)-(c). In this case, there is a distinct difference in performance between the sparsity class and the saturation class of AFs. ReLU and leaky ReLU perform best in terms of training, with a slight advantage over ELU. However, this difference is less prominent in the test classification errors, where ELU is competitive with the rest of its class. A similar clustering occurs between Tanh and Softsign for the saturation class. Both perform very similarly in both training and test errors, with only a slight advantage belonging to Tanh. The Sigmoid AF is convincingly the worst performer of the considered AFs. Though training is slow and stable, the test losses are considerably noisier than those of the remaining AFs.



(a) Standard (b) Standard (c) Standard

(d) No Batch-Norm (e) No Batch-Norm (f) No Batch-Norm

Figure 6.8: (a) Log training error, (b) log test error and (c) the log of step sizes for the CIFAR10 Dataset with the ResNet18 architecture trained using GOLS-I SGD. The standard ResNet architecture includes batch-norm layers, which are omitted in (d)-(f) in order to highlight the effect of skip connections on training with sparsity enforcing ReLU activation functions.

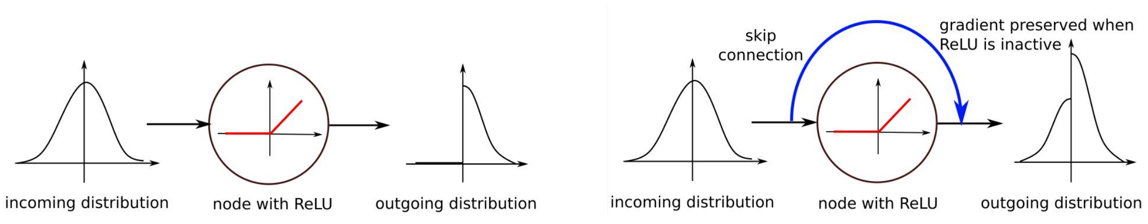
Using Xavier initialization [Glorot and Bengio, 2010], the initial weights for the Sigmoid AF are distributed around 0 in the input domain. However, an input distribution around zero corresponds to a function value output centred around 0.5 for Sigmoid. This is in contrast to the remaining saturation class AFs, which have outputs centred around 0. As shown by Glorot and Bengio [2010], the cumulative effect of 0.5 outputs over a number of hidden layers can push Sigmoid AFs in later layers into saturation, where the derivative is significantly decreased. Batch normalization counters this problem by re-centring the outputs of network layers around 0, where the derivative is at a maximum. Additionally, the maximum derivative of Sigmoid is 0.25, which over many layers diminishes the gradient during backpropagation due to the chain-rule. Batch normalization also compensates for this property by scaling the variance to be 1 for every layer, ensuring that the gradient magnitudes remain adequately scaled. It is clear, that batch normalization has to do a significant amount of "correcting" for the Sigmoid AF in ResNet18. We suspect that the combination of these factors leads GOLS-I to estimate small initial step sizes for Sigmoid, with slow subsequent step size growth. However, the step sizes gradually increase to the point where they are comparable to those of the other AFs after 20,000 iterations.

Apart from the step sizes of the Sigmoid AF, the step size magnitudes between the remaining AFs are similar and approximately constant. This matches the trend observed in Section 6.5.3 for NetII with batch normalization. However, unlike the training performance in the NetII analysis, the Tanh AF does not diverge with ResNet18 and batch normalization. Interestingly, the magnitude and variance of determined step sizes for all AFs (except Sigmoid) increases significantly after 20,000 iterations. This correlates to an increase in gradient variance between data points associated with large losses (resulting in larger gradient norms), and those with lower losses (smaller gradient norms), as the architecture increasingly fits the data. thus, depending on the data-points in mini-batch, $\mathcal{B}_{n,i}$, the magnitude and direction of $\tilde{\boldsymbol{g}}(\boldsymbol{x})$ may vary significantly. Subsequently, this variance is transferred to the directional derivatives used by GOLS-I, leading to a higher variance in step sizes. A plausible corrective measure to manage this variance, is to gradually increase $\mathcal{B}_{n,i}$ as training progresses [Friedlander and Schmidt, 2011, Smith et al., 2017].

Since ResNet is constructed with both skip-connections and batch normalization, it is unclear, which of the two architectural features contribute more significantly towards improving training performance with ReLU. We have considered the contribution of batch normalization to improving training in Section 6.5.3. Here, we consider skip connections more closely. The difference between standard feedforward nodes and skip connected nodes is illustrated in Figure 6.9 with the ReLU AF. A standard node, shown in Figure 6.9(a) passes the incoming distribution through the activation function at the node, which augments the distribution according to its characteristics. As demonstrated in Figure 6.7, this can be problematic with the ReLU AF if poor updates occur, as the propagation of gradients can become obstructed. For a skip connected node, shown in Figure 6.9(b), the incoming distribution is added to the output of the standard node. This ensures, that even in the worst case, where no information passes through the AF, the incoming distribution always propagates forward. Subsequently, evaluated gradients will always be dense.

In investigation 11 we observe the influence of skip connections in ResNet18. Therefore, investigation 11 repeats the analysis of investigation 10, albeit without the use of batch normalization. The results are shown in Figures 6.8(d)-(f). As is consistent with investigations 7 and 9 performed in Section 6.5.3, training performance slows without the use of batch normalization for all AFs. The results of investigation 11 show that training progresses for all AFs, with the exception of the Sigmoid AF. We postulate that this drop in performance is due to scaling difficulties in deep neural networks driven by Sigmoid's positive offset, small maximum derivative and saturating nature [Glorot and Bengio, 2010], which are subsequently not corrected by batch normalization.

We confirmed this phenomenon by conducting numerous additional training runs with modified versions of AFs considered in this analysis. A modified version of the Tanh AF, namely 0.5∗Tanh+0.5 which is centred around 0.5 and has a maximum derivative of 0.5, performed the same as Sigmoid. Conversely, successful training was observed using 2∗Sigmoid−1, which also has a maximum derivative of 0.5 while being centred around 0. This suggests that passing through the origin is an important AF property for promoting effective training with deep networks, an assumption held for both Xavier [Glorot and Bengio, 2010] and He [He et al., 2015] initialization strategies. Additionally, a modified leaky ReLU with maximum output derivative < 0.5 also failed to train. Indeed, Xavier initialization assumes that AF derivatives are 1 around the zero input domain [Glorot and Bengio, 2010]. As Sigmoid satisfies neither of these properties, it is not surprising that its implementation in ResNet18 without batch normalization failed to train successfully.



(a) Standard Node (b) Node with skip-connection

Figure 6.9: Illustration of the structural difference between (a) standard feedforward nodes, and (b) skip-connected nodes.

Importantly, training ResNet18 without bath normalization with the ReLU AF using GOLS-I SGD is not only stable, it also shares the best training performance with leaky ReLU. This is significant, as it demonstrates the effectiveness of skip connections in ensuring that gradients remain dense with ReLU AFs. This allows ReLU's performance to be directly compared to that of the remaining AFs without batch-normalization. Both ReLU and leaky ReLU are outperformed by ELU during the first half of training, but overtake it during the latter half. Figures 6.8(d) and (e) suggest that within the context of skip connections, the difference in performance between ReLU and Leaky ReLU is insignificant.

It is an emerging trend, that the sparsity class of AFs gradually begins to outperform the saturation class as the size of considered neural networks increases. For the foundational problems of Section 6.5.1, this difference is marginal, as the number of nodes in the architecture increases. For the NetII problem in Section 6.5.3, two of the top three performers are from the sparsity class for architectures with and without batch normalization. In the ResNet18 architecture, all of the sparsity class AFs outperform all of the saturation class AFs both with and without batch normalization. This suggests, that the sparsity property becomes increasingly useful, as the size of the network increases. This is consistent with how sparsity operates, as the number of "channels" available to construct the mapping between input and output spaces increases with growing architecture size.

This investigation demonstrates that skip-connections are effective in ensuring that $\tilde{\boldsymbol{g}}(\boldsymbol{x})$ remains dense for ReLU AFs, where sparsity is enforced. In cases where the outputs of ReLU nodes are zero, it is only the residual that is zero, while the information of previous nodes is still passed to subsequent layers through skip connections. This drastically improves GOLS-I SGD's ability to train ReLU neural network architectures. Subsequently, batch normalization layers contribute additional benefit for all AFs considered in our investigations, by increasing robustness and accelerating training. Additionally, this investigation supports the trends observed in Sections 6.5.1 and 6.5.3, where the training performance of larger architectures using GOLS-I SGD is improved by implementing sparsity class AFs.

6.6 Conclusion

In this chapter, we consider the interaction between gradient-only line searches and a variety of neural networks constructed with six different activation functions. The activation functions considered are split into two classes, namely the saturation class (Sigmoid, Tanh and Softsign), and the sparsity class (ReLU, leaky ReLU and ELU). Gradient-only line searches are used to determine the step sizes for gradient based optimizers in full-batch and dynamic mini-batch subsampled (MBSS) loss functions. In our investigations we implement the gradient-only line search that is inexact (GOLS-I) as well as the Gradient-Only Line Search with Bisection (GOLS-B) [Kafka and Wilke, 2019b]. Eleven investigations are conducted using 13 different datasets with a total of 37 network architectures, some using the cross entropy loss and others the mean squared error loss. Training problems include 11 foundational datasets with feedforward neural networks, the MNIST dataset with NetII [Mahsereci and Hennig, 2017] and the CIFAR10 dataset with ResNet18 [He et al., 2016]. These problems cover a range of architectural features, including batch normalization [Ioffe and Szegedy, 2015] and skip connections.

We find, that GOLS is effective in determining the step sizes in dynamic MBSS training for all but a few combinations of activation functions and network architectures. The small neural networks show a close grouping in training performance between the considered activation functions, with a slight advantage belonging to non-linear, saturation type activations. However, training performance of feedforward networks with the ReLU activation function, coupled with GOLS performed poorly. Analyses with NetII and ResNet18 without batch normalization show that a particular activation function can significantly improve the training performance for a given network architecture with GOLS-I. For NetII, the best performers were leaky ReLU and Sigmoid activation functions, while the troublesome performance of ReLU with GOLS-I seen in the foundational problems recurred for NetII.

Our investigations suggest that the predominant cause of GOLS-I's inability to train ReLU architectures are weight updates with step sizes that are too large. These updates shift the distribution of information entering a ReLU activation function fully into its inactive domain, thus producing zero-outputs and halting the flow of information through a node. This can lead to large portions of the network becoming and remaining inactive, which leads to the gradient vector being sparse in these cases. In addition, the implication that there exist domains in the loss function that have zero-gradients means that ReLU loss functions with feedforward architectures can break the assumptions of Lyapunov's global stability theorem, which govern the convergence properties of GOLS-I.

However, the training difficulties encountered with ReLU architectures using GOLS-I can be alleviated by implementation of batch normalization, and skip connections as used in residual networks. Batch normalization centres the distribution of information flowing between sequential network layers around zero, increasing the number of active nodes in the network. Alternatively, skip connections by design guarantee the propagation of input information throughout the network architecture. These technologies ensure that the gradient vector remains dense, allowing GOLS-I to recover form spurious updates, when they occur. Both skip connections and batch normalization render ReLU's competitive with the other activation functions. Batch normalization has the added benefit of accelerating training for all activation functions. Training ResNet18 using GOLS-I with and without batch normalization demonstrated, that additionally implementing batch normalization with skip connections results in a double benefit of stability for ReLU, as well as accelerated training for all activation functions.

Gradient-only line searches are effective at determining adaptive step sizes for gradient descent based training algorithms. Our studies have demonstrated, that the interaction between activation functions and neural network architectures matter. The ResNet18 training problem showed a clear distinction between saturation and sparsity classes of activation functions, with superior training performance belonging to the latter group. The properties of an activation function's derivative have a direct effect on the nature of the loss function presented to the optimization algorithm. Our studies suggest, that activation functions that promote sparsity are better suited to larger network architectures than classical saturation type activation functions. Additionally, significant difficulties can be encountered when training feedforward ReLU architectures with GOLS-I. Therefore, we suggest that practitioners consider technologies such as batch normalization and skip connections, when constructing neural network architectures to be trained with GOLS-I.

Chapter 7

Conclusions and future work

We started by visually exploring the difference between static and dynamic mini-batch subsampling (MBSS) to construct loss function approximations for neural networks. We showed that static MBSS produces smooth, albeit biased, loss surfaces due to sampling errors resulting from the omission of training data in loss evaluations. Conversely, dynamic MBSS trades the bias of individual mini-batches for variance resulting from different sampling errors at every function evaluation. For a training dataset of fixed size and constant mini-batch size, the number of different sampling errors attainable are deterministically limited. This means that dynamic MBSS losses are strictly speaking not noisy and randomly unstructured, but deterministically point-wise discontinuous. Stochasticity is subsequently introduced by the act of randomly selecting mini-batches.

In dynamic MBSS loss landscapes, we explored the ability of local minima and stochastic non-negative associated gradient projection points (SNN-GPPs) to approximate the location of full-batch optima. We showed that SNN-GPPs are spatially isolated around full-batch optima, while local minimizers uniformly identify discontinuities as spurious candidate solutions over the sampled domains. These false local minima remain spread throughout the loss landscapes, even for large mini-batches. We used static MBSS losses to better understand that SNN-GPPs are unlikely to represent the optimum of any individual mini-batch, but rather a solution that is contained within a ball, B_ϵ, that encompasses the optima of all static mini-batches for a given dataset-mini-batch size combination. As the mini-batch size increases, the sampling error decreases, which also reduces the size of B_ϵ. In the limit case of full-batch sampling, the SNN-GPP generalizes to the full-batch optimum, as B_ϵ reduces to a single point.

Subsequently, gradient-only line searches (GOLS) can be constructed to find SNN-GPPs in the form of directional derivative sign changes from negative to positive along a search direction. This affords GOLS the ability to determine step sizes for training algorithms in dynamic MBSS loss functions. We empirically studied the performance of GOLS on an extensive set of training problems, spanning 23 datasets with sizes varying between 150 and 70 000 observations, input dimensions ranging between 4 and 3072 and output dimensions between 2 and 29. Network architectures considered ranged from feedforward neural networks with 1 hidden layer, to skip-connected convolutional deep neural networks with 18 layers and batch normalization. We also optimized a representative training problem of the generative model family in the form of a Variational Autoencoder. These problems encompassed a range of elemental loss function formulations, specifically, the mean squared error (MSE), binary cross entropy (BCE) and KL divergence losses. Activation functions used in various problems included two distinct groups of activation functions, namely the saturation class: Sigmoid, Tanh and Softsign; as well as the sparsity class: ReLU, leaky ReLU and ELU.

Various subsets of these problems were used for different investigations. First, GOLS were extensively compared, and shown to be superior to two minimization line searches, Armijo's rule line search and Golden Section, in dynamic MBSS loss functions. Experiments also showed, that GOLS are able to resolve learning rate schedules that are infeasible to determine a priori. The fruits of these investigations were four different formulations of GOLS, namely GOLS-B, GOLS-

I, GOLS-Back and GOLS-Max. Subsequently, the best overall performer, the *Gradient-Only Line Search that is Inexact* (GOLS-I) was demonstrated to perform in a complementary manner to the Probabilistic Line Search (PrLS). For very small mini-batch sizes ($|\mathcal{B}| = 10$), GOLS-I was outperformed by PrLS. However, at such small mini-batches training is slow overall. As mini-batches increased ($|\mathcal{B}| \geq 50$), which is more representative of mini-batch sizes used in practice, GOLS-I showed a clear performance advantage over PrLS. Additionally, GOLS-I is computationally efficient, as it does not require loss and gradient variance estimates, nor the construction of statistical surrogates.

Having established GOLS-I's ability to determine step sizes effectively in dynamic MBSS loss landscapes, and using the property that SNN-GPPs generalize to smooth functions, it is possible to directly compare static to dynamic MBSS in terms of training performance using the same method. Such analyses demonstrated that dynamic MBSS GOLS-I outperformed static MBSS GOLS-I in training with regards to computational cost, confirming the arguments made by [Bottou, 2010].

Subsequently, we exposed GOLS to a wide range of applications in order to explore their capabilities and potential limitations. We modified the formulations of a number of popular training algorithms to include a line search, namely: Stochastic Gradient Descent, Stochastic Gradient Descent with Momentum, Nesterov's Accelerated Gradient descent, Adagrad, Adadelta, Adam. We also implement GOLS in LBFGS, which naturally accommodates a line search method. Implementing GOLS-I to determine the step sizes for these algorithms revealed, that GOLS-I is capable of automatically adjusting to the properties of the given training algorithms and is competitive with manually tuned fixed step sizes for these algorithms. This investigation also showed, that GOLS-I underperforms with training algorithms that include momentum-like behaviour. However, if this behaviour can be omitted, as is the case for Adam with $\beta_1 = 0$, training performance with GOLS-I is restored.

Finally, we investigated the compatibility of GOLS with various activation functions in different neural network architectures. The continuity and smoothness characteristics of activation functions directly affect the continuity and smoothness characteristics of the loss function and its gradient. Therefore, it is not self-evident that GOLS are compatible with all activation functions. Overall, GOLS were shown to adapt automatically to the properties of saturation and sparsity class activation functions. However, the fact that the ReLU activation function has outputs that are analytically zero over the negative input domain, leads to loss function characteristics that break the assumptions of Lyapunov's global stability theorem. As this theorem governs the convergence properties of GOLS, effective training was not attainable with ReLU in standard feedforward networks. However, these difficulties can be alleviated by implementing network architecture features such as batch normalization and skip connections.

This work has shown, that GOLS present a generalized strategy to automatically determine learning rates of different training algorithms in a wide range of training problems. This eliminates the need for conducting extensive hyper-parameter tuning studies to determine fixed step sizes or parameterize learning rate schedules. Additionally, GOLS offer a number of avenues for continued research.

7.1 Recommendations for future work

Having developed implementations of GOLS in PyTorch 1.0, the immediate next step is to assemble these methods into a single, easily implemented PyTorch optimizer class. This would allow all GOLS methods to be used seamlessly in existing PyTorch models, such that users can explore the technology, as well as extend it. Since GOLS can be used with a variety of training algorithms, it would be of interest to explore any emergent properties stemming from the combination of different GOLS formulations and the different algorithm formulations on a variety of training problems.

Throughout the studies conducted in this thesis, a number of interesting concepts have arisen that are worthy of further consideration. One aspect of particular importance is more extensive comparison between static and dynamic MBSS in line searches. Although we have anecdotally demonstrated that dynamic MBSS line searches can be advantageous over static MBSS line searches, these analyses are by no means exhaustive. Now that it is possible to conduct both static and dynamic MBSS line searches using the same method, the emphasis can shift from line searches to the merits and characteristics of the different sub-sampling approaches. Some preliminary analyses seemed to indicate that conducting static MBSS in SGD produces minima that are close to the furthest edge of B_ϵ, i.e. the bound of B_ϵ which is a farther from starting position $\boldsymbol{x}_n$. Such phenomena need to be more closely studied and qualified.

It is also of interest to quantify the size of B_ϵ in more detail, particularly as a function of training progress. This is independent of how existing GOLS formulations locate SNN-GPPs. Early tests suggest that B_ϵ grows during training, which is intuitive, since algorithms may tend towards areas in the loss with lower curvature as training progresses. For a constant variance, this leads to a larger spread in SNN-GPPs, see Chapter 2. This is a noteworthy consideration when implementing an exact GOLS method such as GOLS-B, as it always finds solutions within B_ϵ. Increasing the mini-batch size during training is one approach to reduce the size of B_ϵ. However, the effect of ball size on training and the rate at which mini-batch size is most effectively increased remains unclear.

Conversely, training results in Chapter 6 suggested that the larger size of B_ϵ together with a conservative bracketing strategy for GOLS-B might act as a regularizer in training. Conservative GOLS-B naturally shrinks steps sizes as the algorithm approaches an optimum. This behaviour is consistent with a number of predetermined learning rate schedules, such as simulated annealing [Kirkpatrick et al., 1983]. Reducing step sizes in a manner that is in accordance with the characteristics of loss functions might be a means by which to manage and delay overfitting.

Finally, all of the popular training algorithms considered in this study make use of gradient information to construct their search directions. However, the gradient is also sensitive to the sampling error introduced by conducting MBSS. It is therefore of interest to investigate the interaction between MBSS and search direction quality. It is conceivable, that the descent directions of Stochastic Gradient Descent (SGD) exhibit a form of conjugacy due to sampling error. Benefit can also be derived from overshooting along SGD descent directions, even in the case where full-batch sampling is used to resolve step sizes along the given search direction. The Variational Autoencoder analysis in Chapter 3 showed indications of this phenomenon. Therefore, it would be of interest to qualitatively assess this phenomenon. Subsequently, it would be possible to uncouple the effect of search direction quality, from the ability to resolve SNN-GPPs along a given search direction, on training performance. It is unclear at this point, whether the search direction itself, or the accuracy of the resolved optimum is more affected by dynamic MBSS, and which is more detrimental to training performance.

Provided continued analyses find dynamic MBSS line searches to be superior to those of static MBSS, there are also questions concerning how dynamic MBSS and GOLS might affect other algorithms from the field of mathematical programming. Although we include LBFGS with GOLS in our investigations, the interaction between stochastic line searches and second-order methods (including stochastic variants [Hennig, 2013]) needs to be further studied to explore the potential benefit of approximate Hessians.

The investigations performed in Chapters 3 (Section 3.4.5) and 6 have demonstrated that GOLS are not only able to automatically determine learning rates, but can also be used as explorative tools in dynamic MBSS loss landscapes. As discussed, our investigations suggest that there are further questions to be asked and other concepts to be examined. We therefore hope this work will find other curious minds along the path towards improved machine learning optimization.

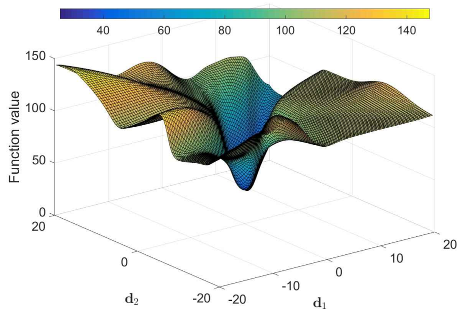
Appendix A

Appendix

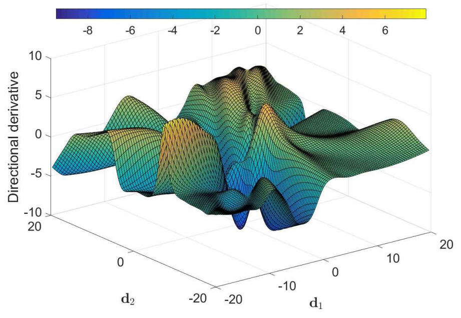
A.1 Figures of Additional Activation Functions from Visual Investigations

Appended here, we show figures relating to the additional activation functions considered in our visual study of Chapter 2. These investigations are divided into global and local domains, spanning steps along two random directions with ranges [-20,20] and [-2,2] respectively.

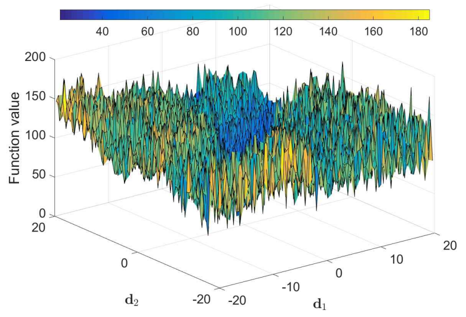
A.1.1 Relating to Global Activation Function Characteristics in Random Directions over [-20,20] Domains in Weight Space



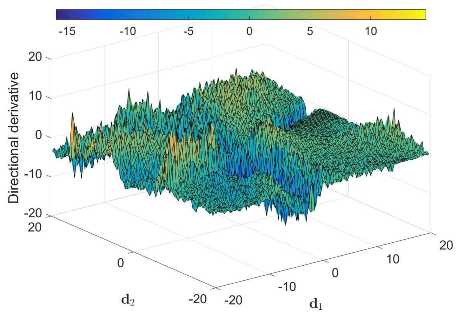
(a) Function value, $M = 150$



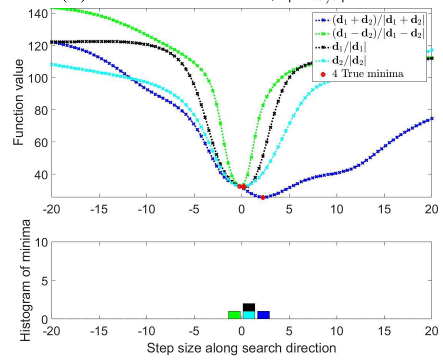
(b) Directional derivative, $M = 150$



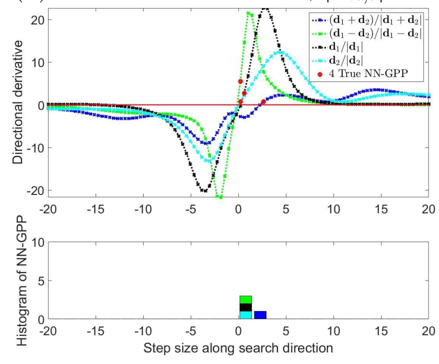
(c) Function value, $|\mathcal{B}_{n,i}| = 10$



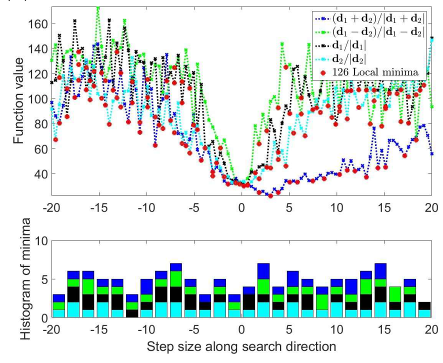
(d) Directional derivative, $|\mathcal{B}_{n,i}| = 10$



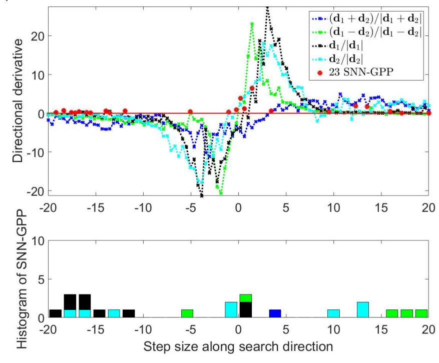
(e) True minima along search directions



(f) True NN-GPPs along search directions

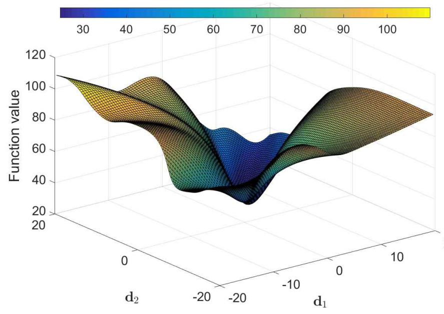


(g) Local minima along search directions

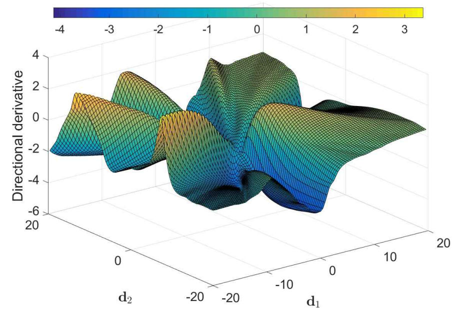


(h) SNN-GPPs along search directions

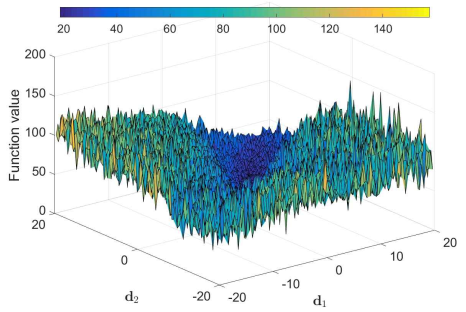
Figure A.1: The Tanh AF: A steeper AF derivative around 0 means more curvature in the loss gradients, which manifest as steeper directional derivatives. This aids SNN-GPP localization in the active region (origin of (h)), but not at saturation (outer ranges of (h)).



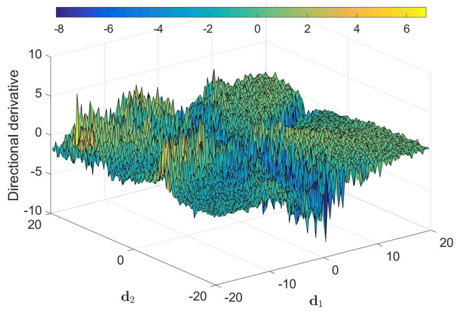
(a) Function value, $M = 150$



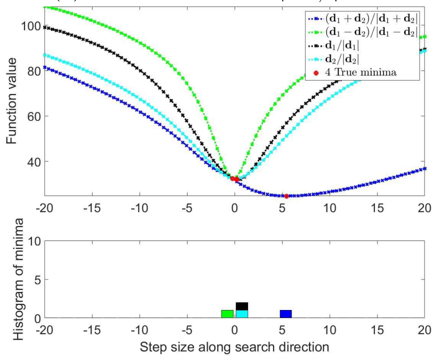
(b) Directional derivative, $M = 150$



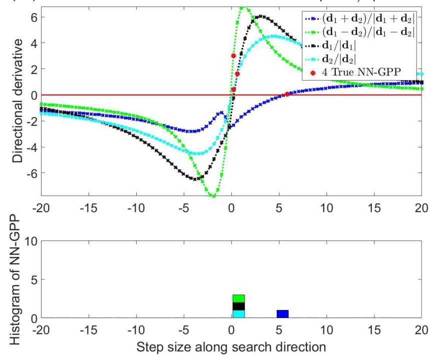
(c) Function value, $|\mathcal{B}_{n,i}| = 10$



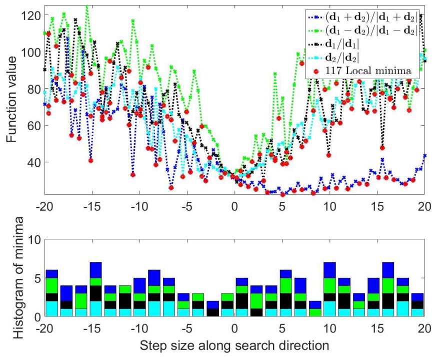
(d) Directional derivative, $|\mathcal{B}_{n,i}| = 10$



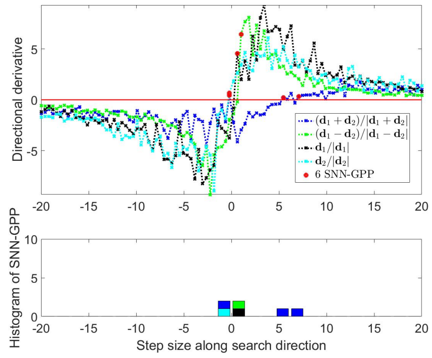
(e) True minima along search directions



(f) True NN-GPPs along search directions

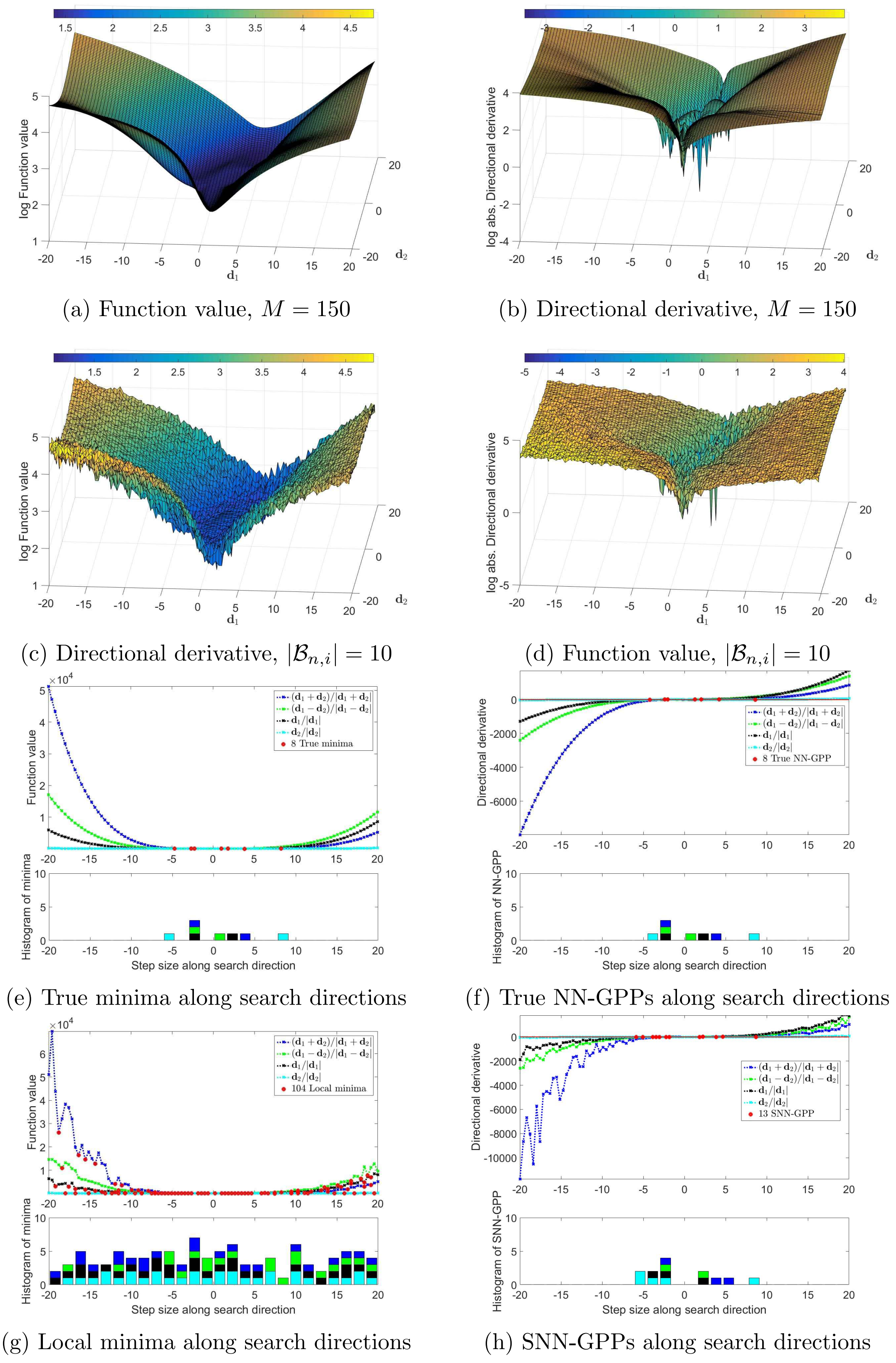


(g) Local minima along search directions



(h) SNN-GPPs along search directions

Figure A.2: The Softsign AF: The less aggressive taper-off in the AF derivative reduces the chance for spurious SNN-GPPs in the saturation regions. In the centre domain, SNN-GPPs are highly localized around the full-batch, true optima.



(a) Function value, $M = 150$

(b) Directional derivative, $M = 150$

(c) Directional derivative, $|\mathcal{B}_{n,i}| = 10$

(d) Function value, $|\mathcal{B}_{n,i}| = 10$

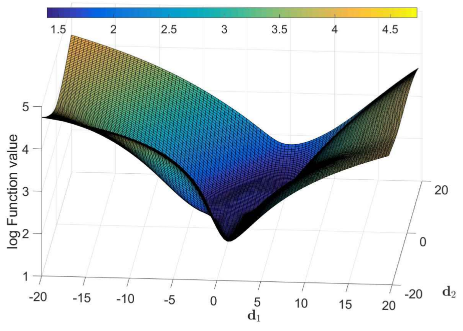
(e) True minima along search directions

(f) True NN-GPPs along search directions

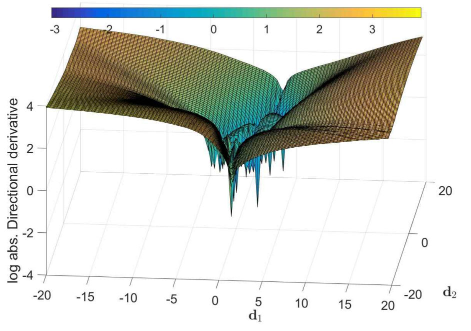
(g) Local minima along search directions

(h) SNN-GPPs along search directions

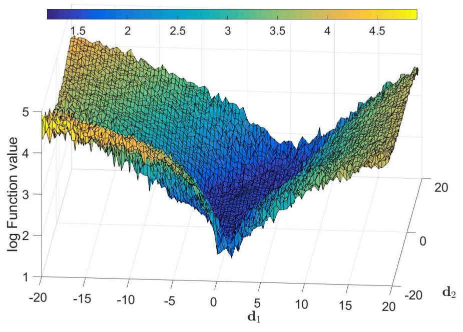
Figure A.3: The ReLU AF: Pushing ReLU activations far into their active domain results in convex behaviour of the MSE loss function on a large scale. However, directional derivative and true optima plots indicate that more detail is contained in the basin.



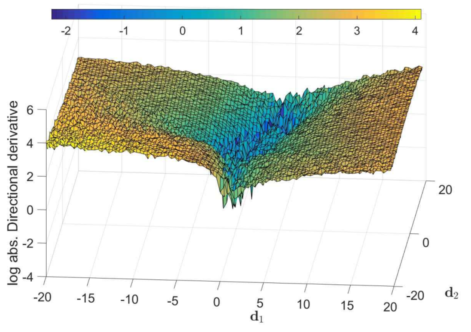
(a) Function value, $M = 150$



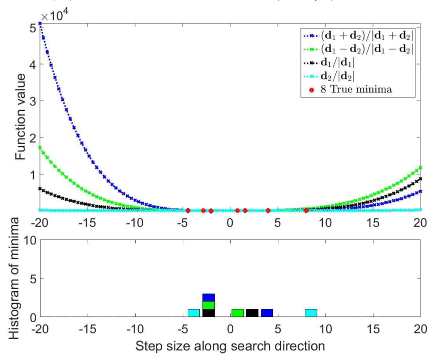
(b) Directional derivative, $M = 150$



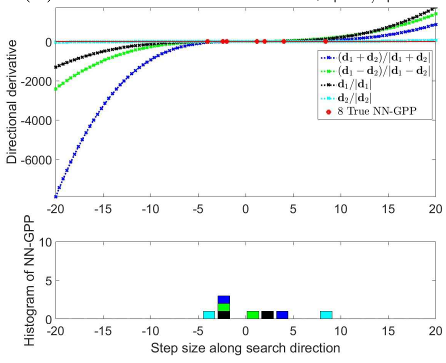
(c) Function value, $|\mathcal{B}_{n,i}| = 10$



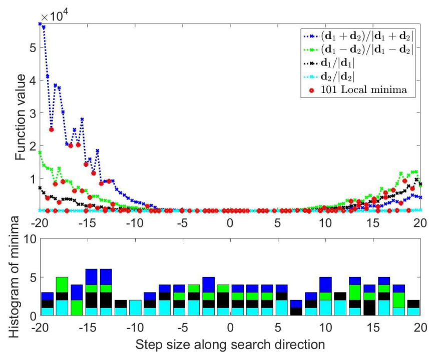
(d) Directional derivative, $|\mathcal{B}_{n,i}| = 10$



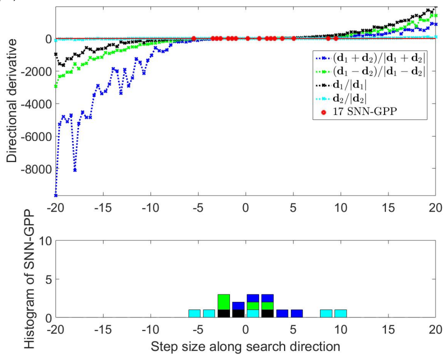
(e) True minima along search directions



(f) True NN-GPPs along search directions

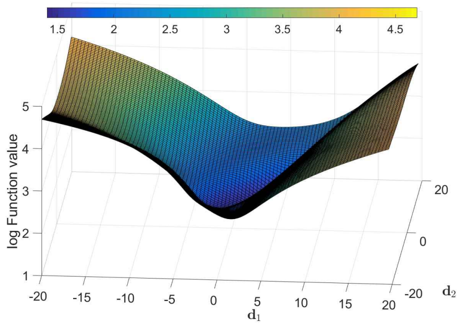


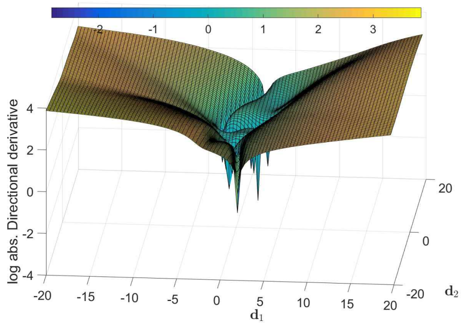
(g) Local minima along search directions

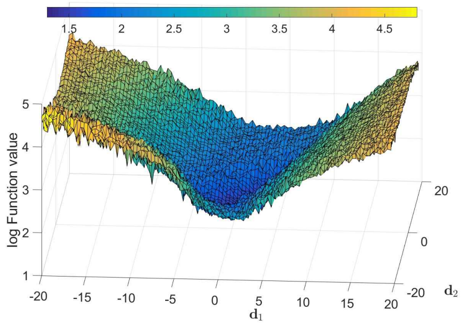


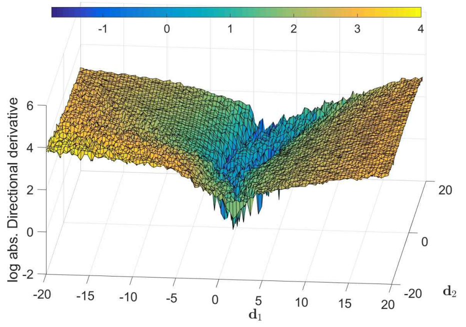
(h) SNN-GPPs along search directions

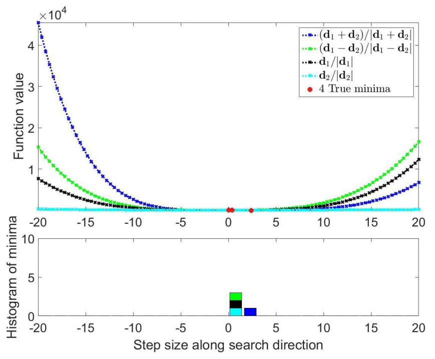
Figure A.4: The leaky ReLU AF: Since the magnitude of the "leaky" gradient is relatively small, its contribution is not apparent at this length scale. Therefore, the plots look very similar to those of ReLU. SNN-GPPs are concentrated around the centre, where the magnitude of the directional derivatives is small.

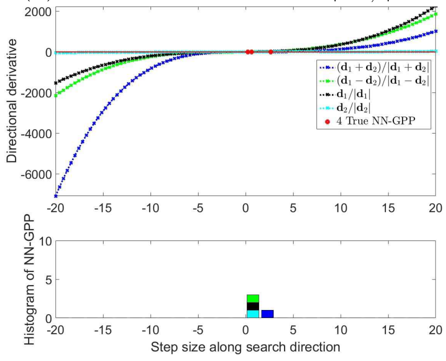

(a) Function value, $M = 150$

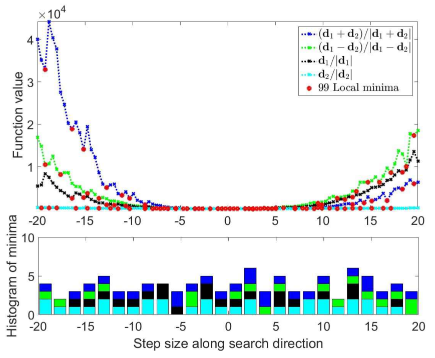

(b) Directional derivative, $M = 150$


(c) Function value, $|\mathcal{B}_{n,i}| = 10$


(d) Directional derivative, $|\mathcal{B}_{n,i}| = 10$


(e) True minima along search directions


(f) True NN-GPPs along search directions


(g) Local minima along search directions

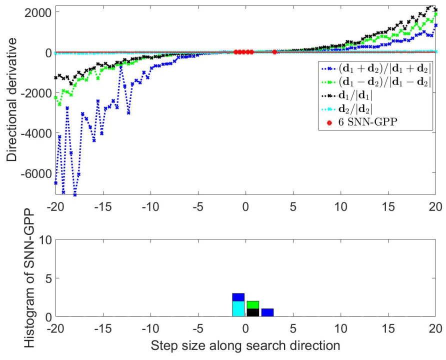
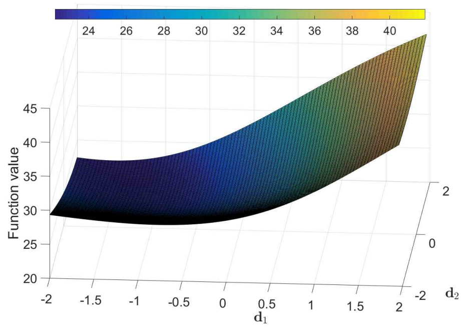
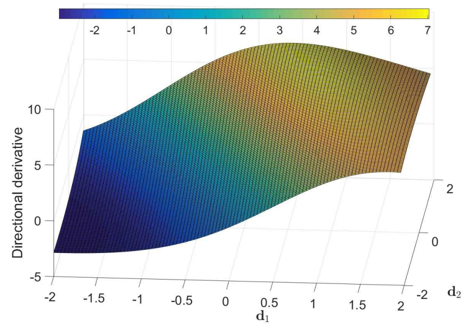

(h) SNN-GPPs along search directions

Figure A.5: The ELU AF: The shapes of the convex element of the loss function are similar to those of the other ReLUs. However, the structure in the basin seems to be different. This is confirmed by the number of true optima. The narrow spatial grouping of SNN-GPPs is the contribution of the continuous AF derivative.

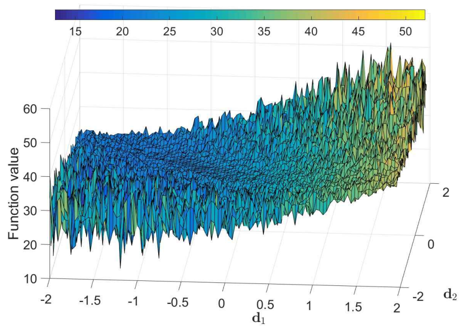
A.1.2 Relating to Local Activation Function Characteristics in Descent Directions over [-2,2] Domains in Weight Space



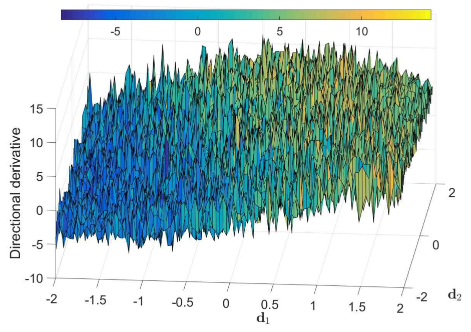
(a) Function value, $M = 150$



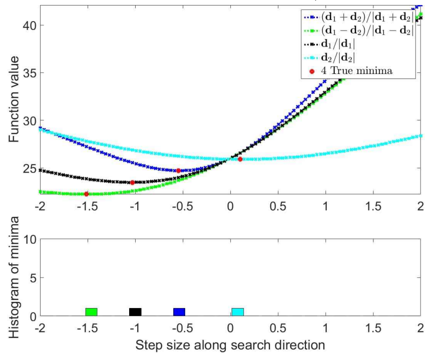
(b) Directional derivative, $M = 150$



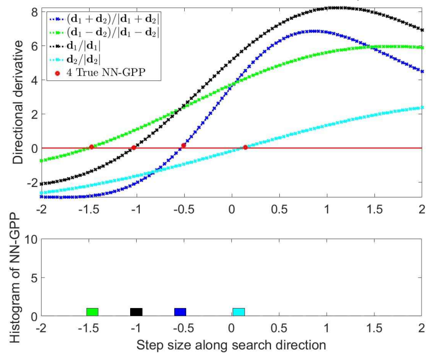
(c) Function value, $|\mathcal{B}_{n,i}| = 10$



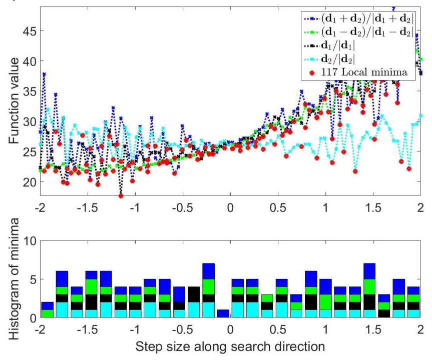
(d) Directional derivative, $|\mathcal{B}_{n,i}| = 10$



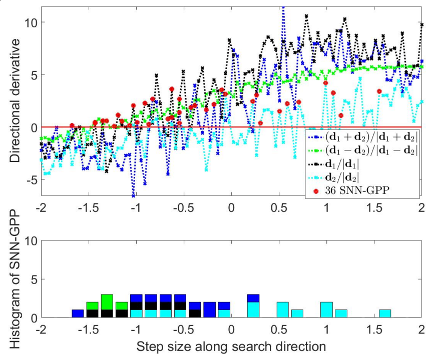
(e) True minima along search directions



(f) True NN-GPPs along search directions

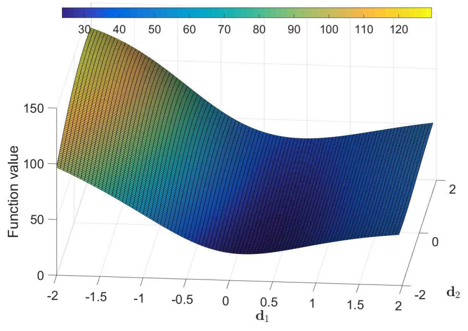


(g) Local minima along search directions

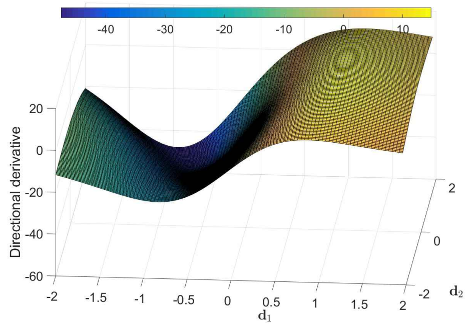


(h) SNN-GPPs along search directions

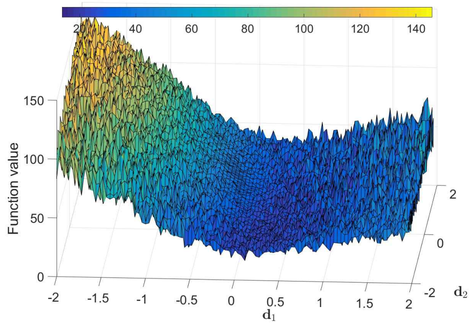
Figure A.6: The Sigmoid AF close-up: Shapes are smooth and have little curvature. SNN-GPPs have a smaller spatial range in directions where the curvature is larger, while being more spread out in low curvature directions.



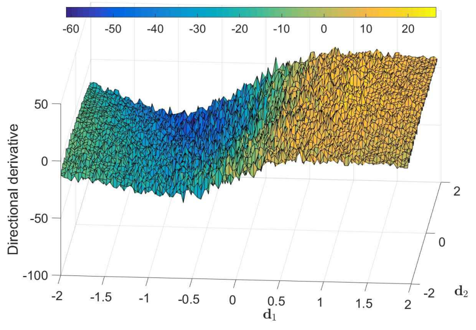
(a) Function value, $M = 150$



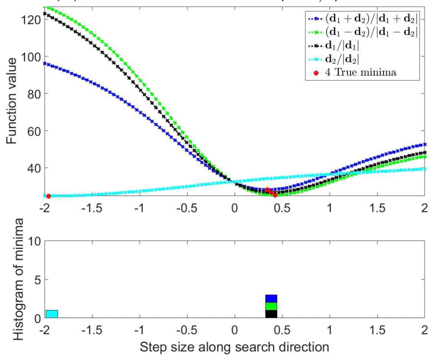
(b) Directional derivative, $M = 150$



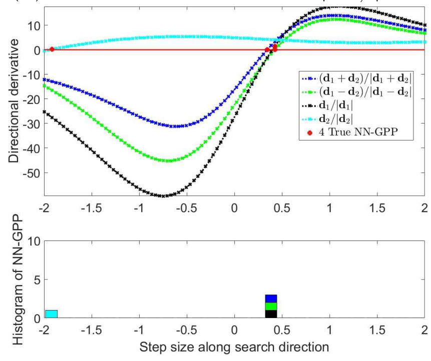
(c) Function value, $|\mathcal{B}_{n,i}| = 10$



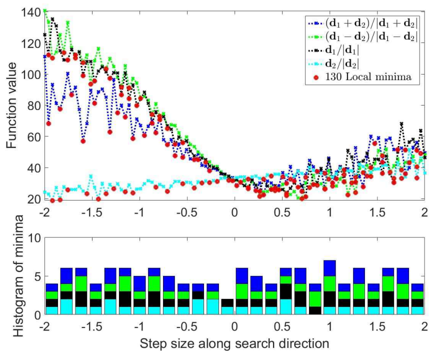
(d) Directional derivative, $|\mathcal{B}_{n,i}| = 10$



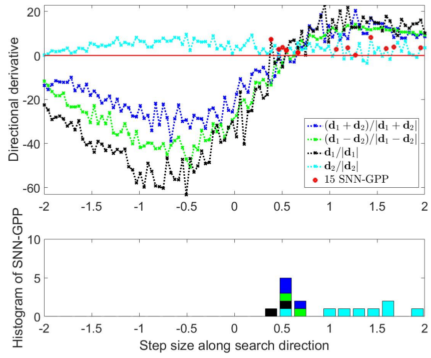
(e) True minima along search directions



(f) True NN-GPPs along search directions

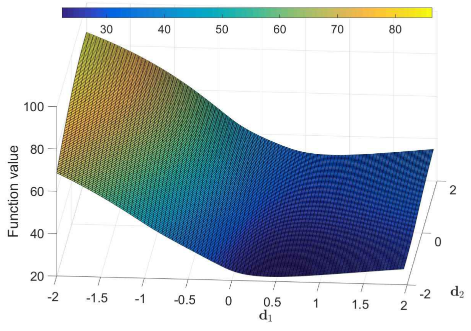


(g) Local minima along search directions

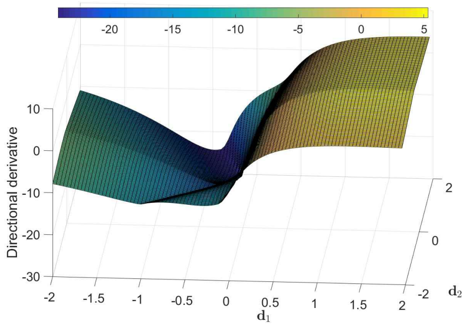


(h) SNN-GPPs along search directions

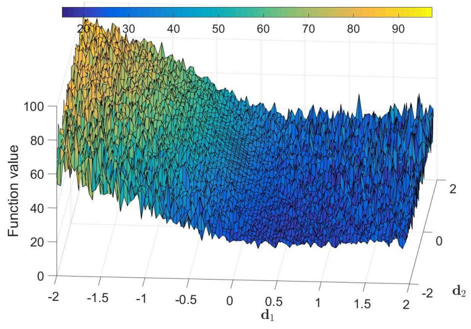
Figure A.7: The Tanh AF close-up: The higher curvature of Tanh helps localize SNN-GPPs. Though this example captures lower variance directions in the function value (see (g), $\boldsymbol{d}_2$), this does not alleviate the problem of uniformly spread spurious local minima.



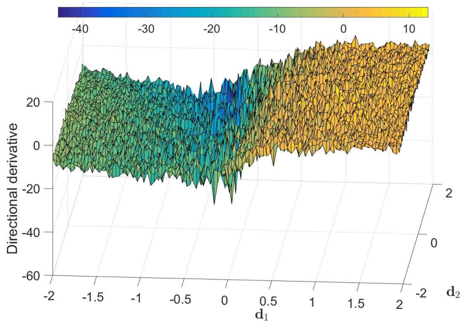
(a) Function value, $M = 150$



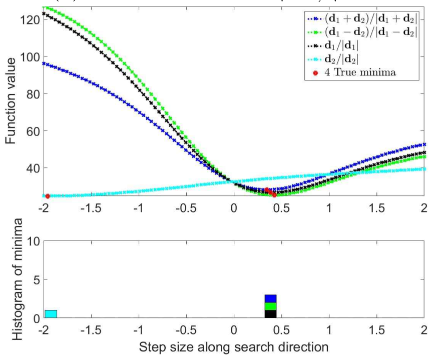
(b) Directional derivative, $M = 150$



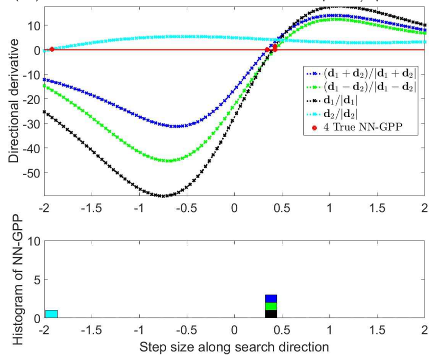
(c) Function value, $|\mathcal{B}_{n,i}| = 10$



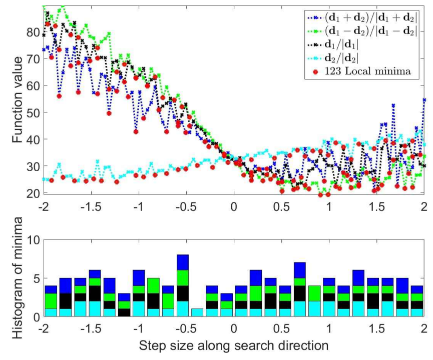
(d) Directional derivative, $|\mathcal{B}_{n,i}| = 10$



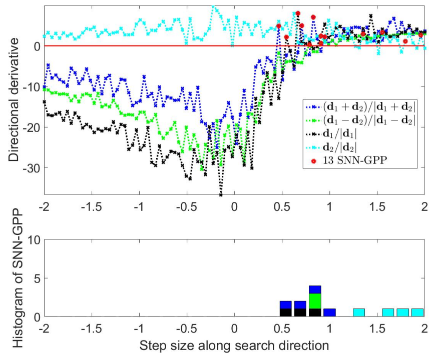
(e) True minima along search directions



(f) True NN-GPPs along search directions

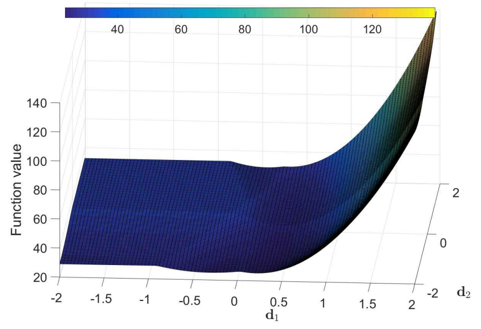


(g) Local minima along search directions

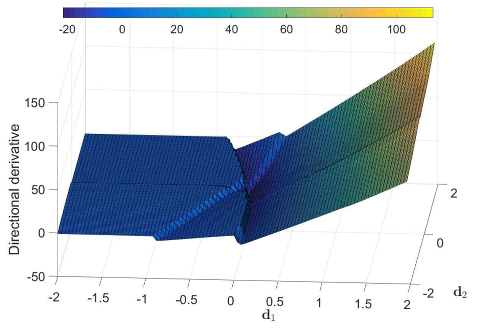


(h) SNN-GPPs along search directions

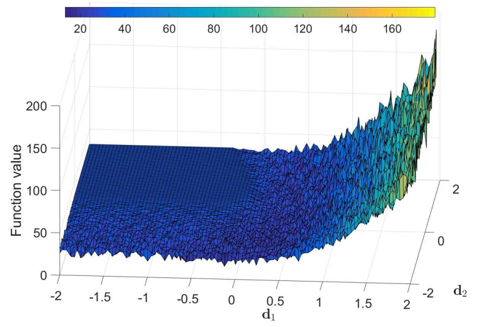
Figure A.8: The Softsign AF close-up: The small domain investigation confirms that the less aggressive taper-off of the AF derivative contributes to localizing SNN-GPPs around the true optimum, while avoiding spurious instances at saturation. Local minima remain uniformly distributed.



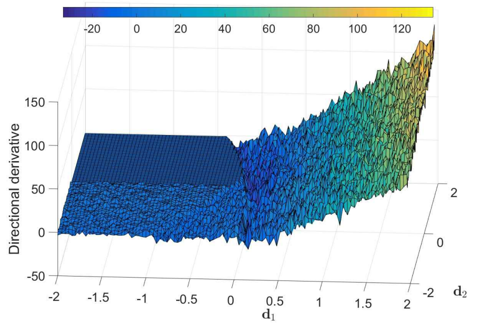
(a) Function value, $M = 150$



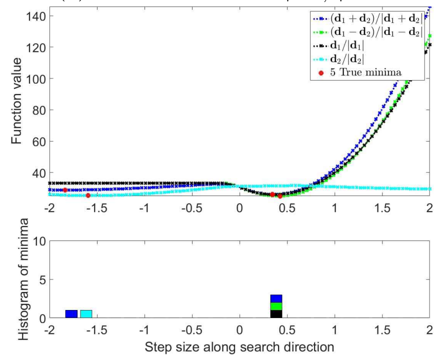
(b) Directional derivative, $M = 150$



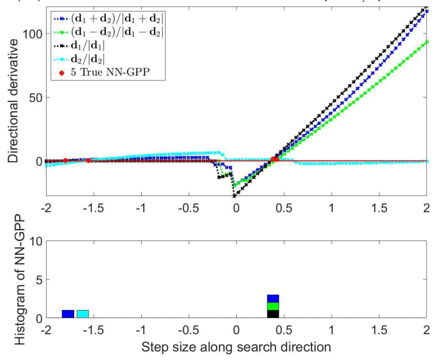
(c) Function value, $|\mathcal{B}_{n,i}| = 10$



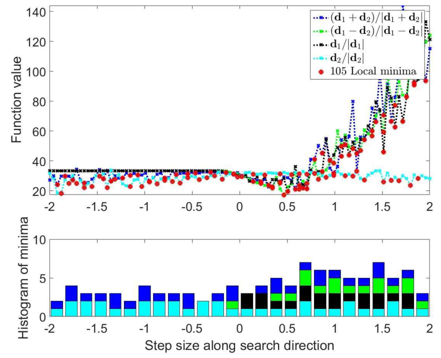
(d) Directional derivative, $|\mathcal{B}_{n,i}| = 10$



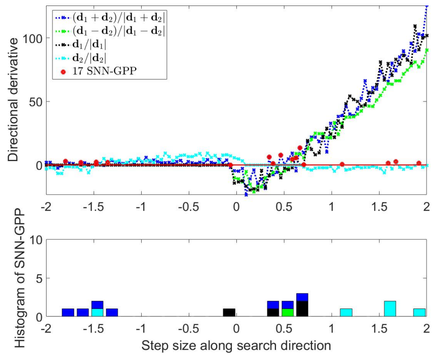
(e) True minima along search directions



(f) True NN-GPPs along search directions

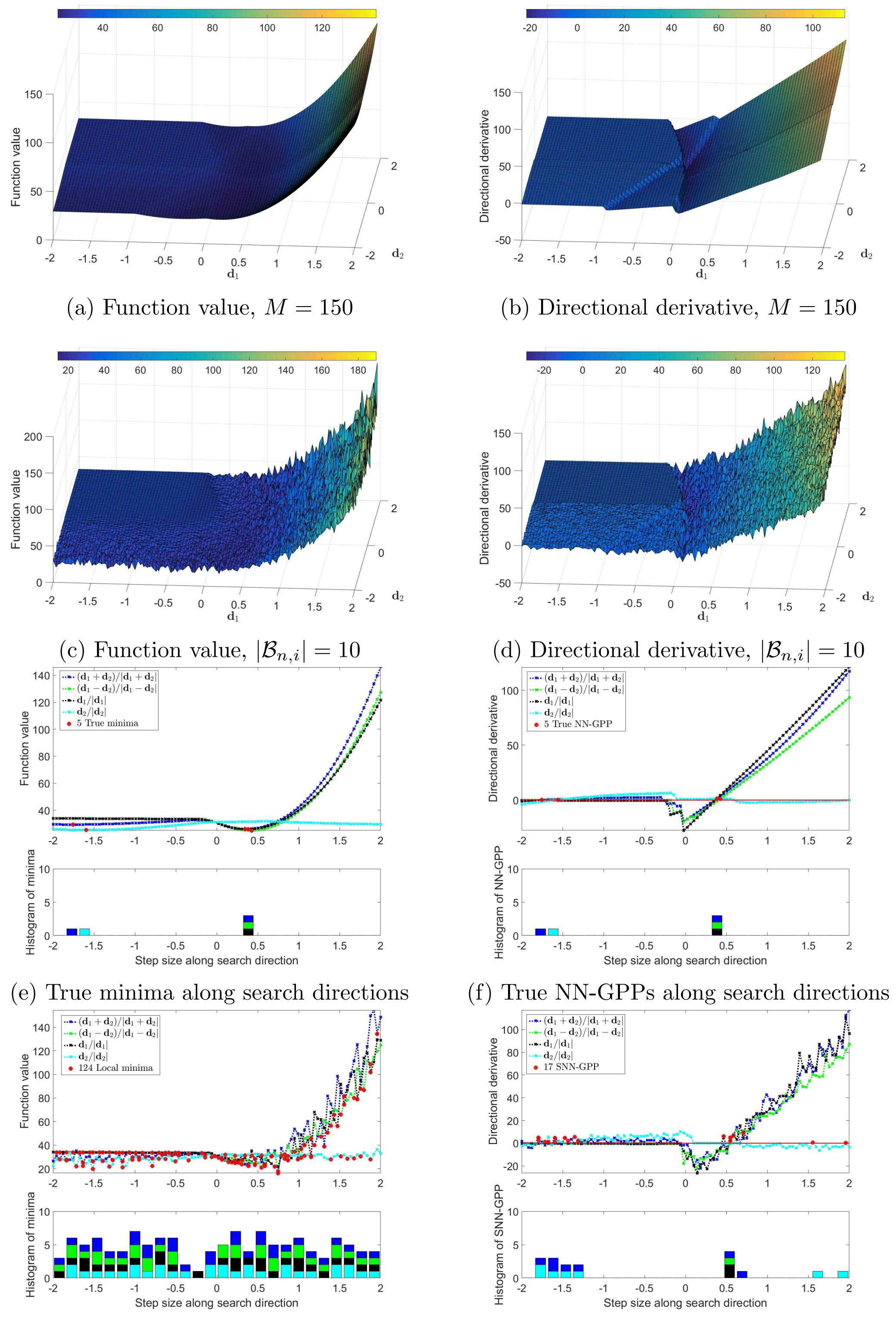


(g) Local minima along search directions



(h) SNN-GPPs along search directions

Figure A.9: The ReLU AF close-up: Notable features are the stark changes in the directional derivatives. These correspond to the "activation" and "deactivation" of various nodes in the network. A unique feature to the ReLU activation is the presence of flat planes where the directional derivative is 0. These areas denote weight spaces where no information passes through the network, for all mini-batches.



(a) Function value, $M = 150$

(b) Directional derivative, $M = 150$

(c) Function value, $|\mathcal{B}_{n,i}| = 10$

(d) Directional derivative, $|\mathcal{B}_{n,i}| = 10$

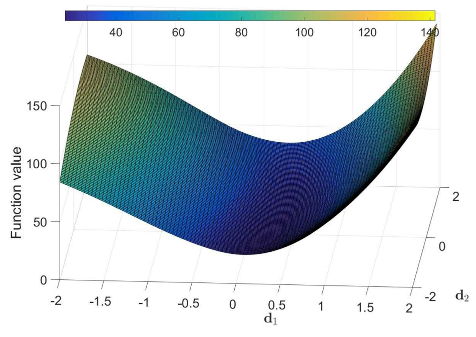
(e) True minima along search directions

(f) True NN-GPPs along search directions

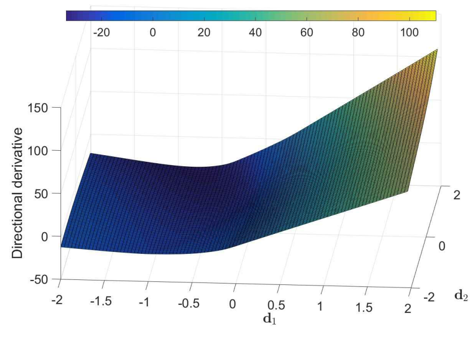
(g) Local minima along search directions

(h) SNN-GPPs along search directions

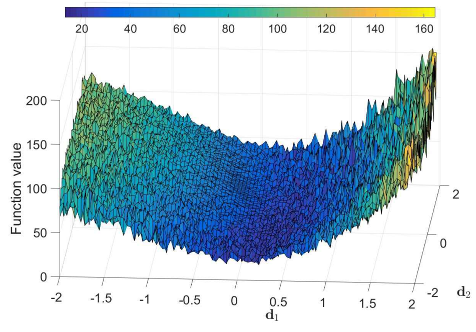
Figure A.10: The leaky ReLU AF close-up: Flat planes with constant directional derivative values are also present here. Although in this case they have a non-zero numerical value, they do not contribute significantly to localizing SNN-GPPs or reducing the number of true optima for the ReLU class.



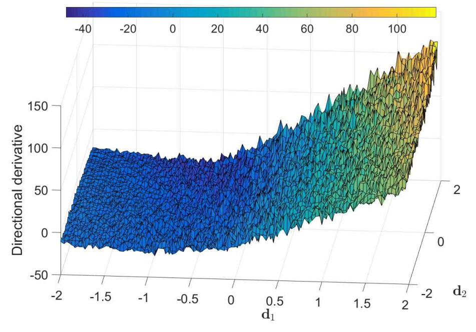
(a) Function value, $M = 150$



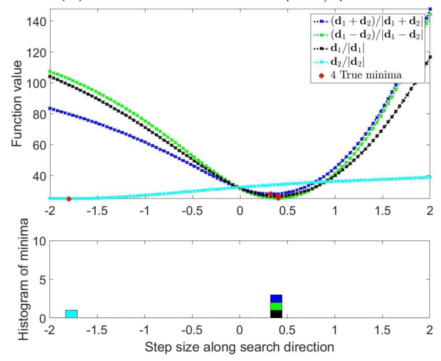
(b) Directional derivative, $M = 150$



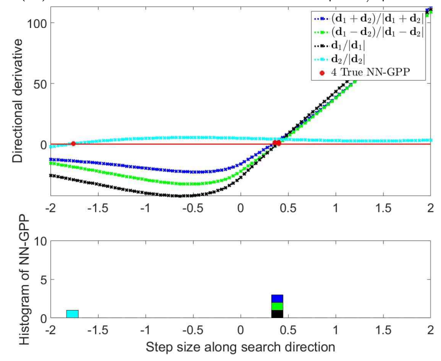
(c) Function value, $|\mathcal{B}_{n,i}| = 10$



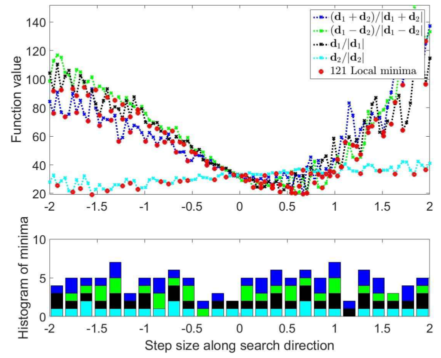
(d) Directional derivative, $|\mathcal{B}_{n,i}| = 10$



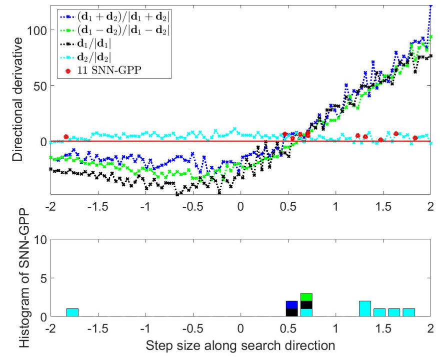
(e) True minima along search directions



(f) True NN-GPPs along search directions

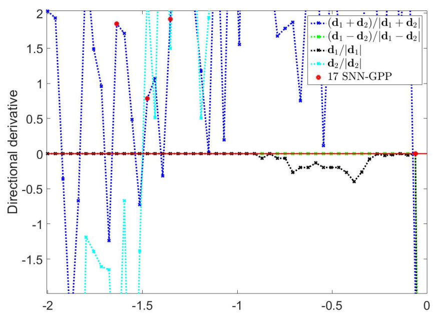


(g) Local minima along search directions

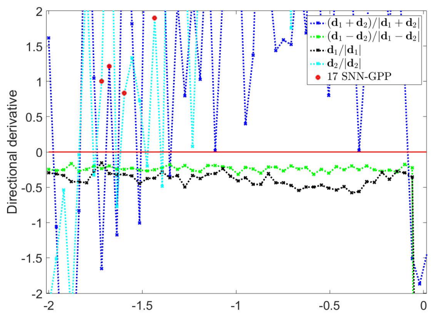


(h) SNN-GPPs along search directions

Figure A.11: The ELU AF close-up: The smooth exponential derivative in the negative domain of the AF results in a greater magnitude in negative directional derivatives in the loss ((d) and (h)) compared to the remainder of the sparsity class. This in turn helps distance the directional derivatives from 0, benefiting localization of SNN-GPPs compared to leaky ReLU.



(a) ReLU



(b) Leaky ReLU

Figure A.12: Detailed comparison of directional derivative plots between ReLU and leaky ReLU activations when hidden units become "inactive". As expected, ReLU units switch "off" entirely, containing no gradient information, while the leaky ReLU results in non-zero directional derivatives.

A.2 Artificial Neural Networks

The single and double hidden layer feedforward neural network architectures are expressed mathematically by Equations (A.1) and (A.2) respectively below. The optimization vector, $\boldsymbol{x}$ is sectioned and transformed into matrices $\boldsymbol{X}^{(c)}$ for the relevant weights in connection layers c of the network. The given data observation pair $\boldsymbol{t}_b$ is separated to give the input data, $\boldsymbol{T}^i_b$, and output data, $\boldsymbol{T}^o_b$. Suppose a given dataset has an input domain, $\boldsymbol{T}^i$ with $|\mathcal{B}|$ observations and D dimensions (features). The respective output domain, $\boldsymbol{T}^o$, has corresponding observations $|\mathcal{B}|$ and output dimensions E (classes). Then for every observation b and every output dimension e, a prediction of the output data $\hat{\boldsymbol{T}}^o$ can be constructed from the original data input domain $\boldsymbol{T}^i$, given by

$$\hat{\boldsymbol{T}}^o_{be} = a_{outer}(\sum_{b=1}^{M_1} \boldsymbol{X}^{(2)}_{ej} a_{inner}(\sum_{i=1}^{D} \boldsymbol{X}^{(1)}_{ji} \boldsymbol{T}^i_{bi} + \boldsymbol{X}^{(1)}_{j0}) + \boldsymbol{X}^{(2)}_{e0}), \tag{A.1}$$

for a single hidden layer neural network and

$$\hat{\boldsymbol{T}}^o_{be} = a_{outer}(\sum_{l=1}^{M_2} \boldsymbol{X}^{(3)}_{le} a^{(2)}_{inner}(\sum_{b=1}^{M_1} \boldsymbol{X}^{(2)}_{ej} a^{(1)}_{inner}(\sum_{i=1}^{D} \boldsymbol{X}^{(1)}_{ji} \boldsymbol{T}^i_{bi} + \boldsymbol{X}^{(1)}_{j0}) + \boldsymbol{X}^{(2)}_{e0}) + \boldsymbol{X}^{(3)}_{l0}), \tag{A.2}$$

for a double hidden layer neural network.

The number of nodes in the respective hidden layers is given by M_n, $n \in [1, 2]$. The nodal activation function is denoted by a and $\boldsymbol{X}^{(c)}$, $c \in [1, 2, 3]$ denotes the set of weights connecting sequential layers in the network between the input layer and the output layer in a forward direction. Thus the single hidden layer network has two sets of weights, $\boldsymbol{X}^{(c)}$, and the double hidden layer network has three respectively [Bishop, 2006].

The nodal weights $\boldsymbol{x}$ are optimized to a configuration which best captures the relationship between the input and output data spaces. The loss-function used is the mean squared error (MSE), determined over every b in batch size $|\boldsymbol{B}|$ and every class $e \in E$ according to the Proben1 dataset guidelines [Prechelt, 1994] as:

$$\ell(\boldsymbol{x}, \boldsymbol{t}_b) = \frac{100}{E \cdot |\boldsymbol{B}|} \sum_{b=1}^{|\boldsymbol{B}|} \sum_{e=1}^{E} (\hat{\boldsymbol{T}}^o_{be}(\boldsymbol{x}) - \boldsymbol{T}^o_{be})^2, \tag{A.3}$$

where $\hat{\boldsymbol{T}}^o(\boldsymbol{x})$ is the output estimation of the current network configuration as a function of the weights, and $\boldsymbol{T}^o$ is the target output of the corresponding training dataset samples.

A.3 Pseudo code for Gradient-Only Line Searches

In this section we present the pseudo code of the gradient-only line searches developed in Chapter 3. We give the pseudo code in terms of symbols that are agnostic to the mini-batch subsampling method used. This is done in order to highlight that both approaches are legitimate and these line searches generalize to all forms of sampling. In the limit case, gradient-only line searches also adapt seamlessly to full-batch sampling.

A.3.1 Exact Line Search: Gradient-Only Line Search with Bisection (GOLS-B)

The directional derivative values used in this method are defined as $F'_n(\alpha) = \boldsymbol{g}(\boldsymbol{x}_n + \alpha \cdot \boldsymbol{d}_n)^T \boldsymbol{d}_n$ and the search direction, $\boldsymbol{d}_n$, at the respective values for α at the different points.

Algorithm 3: GOLS-B: Gradient-Only Line Search with Bisection

```
   Input: $F'_n(\alpha)$, $\boldsymbol{d}_n$
   Output: $\alpha_{n,I_n}$, $I_n$
1  Define constants:Define constants: $\delta = 5$, $r = \frac{\sqrt{5}+1}{2}$, maximum step size $\alpha_{max}$, minimum
     step size $\alpha_{min}$, $tol = 10^{-12}$, flag $= 1$, $i = 0$, $i_{max} = 1000$.
2  Determine step sizes for: the lower bound, $\alpha_n^l = 0$; middle, $\alpha_n^m = \delta$ and upper,
     $\alpha_n^u = \alpha_n^m + r \cdot \delta$
3  flag = 1
4  evaluate $F'_n(\alpha_n^m)$, increment $i$
5  evaluate $F'_n(\alpha_n^u)$, increment $i$
6  if if $\alpha_n^u > \alpha_{max}$ then
7      $\alpha_n^u = \alpha_{max}$
8      $I = \alpha_n^u - \alpha_n^l$
9      $\alpha_n^m = \alpha_n^l + \frac{1}{2}I$
10     evaluate $F'_n(\alpha_n^m)$, increment $i$
11     evaluate $F'_n(\alpha_n^u)$, increment $i$
12 while $F'_n(\alpha_n^u) < 0$ and flag and $i < i_{max}$ do
13     $\alpha_n^m = \alpha_n^u$
14     $\alpha_n^u = \alpha_n^m + r^i \cdot \delta$, where $i$ is the number of function evaluations
15     evaluate $F'_n(\alpha_n^u)$, increment $i$
16     if $\alpha_n^u > \alpha_{max}$ then
17         flag = 0
18         $\alpha_{n,I_n} = \alpha_{max}$
19 if if flag = 1, reduce the interval then
20     Define Interval, $I = \alpha_n^u - \alpha_n^l$
21     while $I > tol$ and $\alpha_n^u > \alpha_{min}$ and $i < i_{max}$ do
22         if $F'_n(\alpha_n^m) < 0$ and $F'_n(\alpha_n^u) > 0$ then
23             $\alpha_n^l = \alpha_n^m$
24             $I = \alpha_n^u - \alpha_n^l$
25         if $F'_n(\alpha_n^m) > 0$ then
26             $\alpha_n^u = \alpha_n^m$
27             $F'_n(\alpha_n^u) = F'_n(\alpha_n^m)$
28             $I = \alpha_n^u - \alpha_n^l$
29         $\alpha_n^m = \alpha_n^l + \frac{1}{2}I$
30         Evaluate the new $F'_n(\alpha_n^m)$, increment $i$
31     finalize the step size: $\alpha_{n,I_n} = \frac{\alpha_n^u + \alpha_n^l}{2}$
```

A.3.2 Inexact Line Search: Gradient-Only Line Search that is Inexact (GOLS-I)

Parameters used for this method are: $\eta = 2$, $c_2 = 0.9$, $\alpha_{min} = 10^{-8}$ and $\alpha_{max} = 10^7$. $F_n'(\alpha) = \boldsymbol{g}(\boldsymbol{x}_n + \alpha \cdot \boldsymbol{d}_n)^T \boldsymbol{d}_n$.

Algorithm 4: GOLS-I: Gradient-Only Line Search that is Inexact

```
   Input: $F_n'(\alpha)$, $\boldsymbol{d}_n$ , $\alpha_{n,0}$
   Output: $\alpha_{n,I_n}$, $I_n$
1  Define constants: $\alpha_{min} = 10^{-8}$, flag = 1, $\eta = 2$, $c_2 = 0.9$, $i = 0$
2  $\alpha_{max} = min(\frac{1}{||\boldsymbol{d}_n||_2}, 10^7)$
3  Evaluate $F_n'(0)$, increment $i$ (or use saved gradient from last $F_{n-1}'(\alpha_{n-1,I_{n-1}})$, to
     evaluate $\boldsymbol{g}(\boldsymbol{x}_{n-1} + \alpha_{n-1,I_{n-1}} \cdot \boldsymbol{d}_{n-1})^T \boldsymbol{d}_n$ without incrementing $i$)
4  if $\alpha_{n,0} > \alpha_{max}$ then
5      $\alpha_{n,0} = \alpha_{max}$
6  if $\alpha_{n,0} < \alpha_{min}$ then
7      $\alpha_{n,0} = \alpha_{min}$
8  Evaluate $F_n'(\alpha_{n,0})$, increment $i$
9  Define $tol_{dd} = |c_2 F_n'(0)|$
10 if $F_n'(\alpha_{n,0}) > 0$ and $\alpha_{n,0} < \alpha_{max}$ then
11     flag = 1, decrease step size
12 if $F_n'(\alpha_{n,0}) < 0$ and $\alpha_{n,0} > \alpha_{min}$ then
13     flag = 2, increase step size
14 if $F_n'(\alpha_{n,0}) > 0$ and $F_n'(\alpha_{n,0}) < tol_{dd}$ then
15     flag = 0, immediate accept condition
16 while flag > 0 do
17     if flag = 2 then
18         $\alpha_{n,i+1} = \alpha_{n,i} \cdot \eta$
19         Evaluate $F_n'(\alpha_{n,i+1})$
20         if $F_n'(\alpha_{n,i+1}) \geq 0$ then
21             flag = 0
22         if $\alpha_{n,i+1} > \frac{\alpha_{max}}{\eta}$ then
23             flag = 0
24     if flag = 1 then
25         $\alpha_{n,i+1} = \frac{\alpha_{n,i}}{\eta}$
26         Evaluate $F_n'(\alpha_{n,i+1})$
27         if $F_n'(\alpha_{n,i+1}) < 0$ then
28             flag = 0
29         if $\alpha_{n,i+1} < \alpha_{min} \cdot \eta$ then
30             flag = 0
31 $\alpha_{n,I_n} = \alpha_{n,i+1}$
```

A.3.3 Inexact Line Search: Gradient-Only Line Search Maximizing step size (GOLS-Max)

Parameters used for this method are: $\eta = 2$, $c_2 = 0.9$, $\alpha_{min} = 10^{-8}$ and $\alpha_{max} = 10^7$. $F_n'(\alpha) = \boldsymbol{g}(\boldsymbol{x}_n + \alpha \cdot \boldsymbol{d}_n)^T \boldsymbol{d}_n$.

Algorithm 5: GOLS-Max: Gradient-Only Line Search Maximizing step size

```
   Input: $F_n'(\alpha)$, $\boldsymbol{d}_n$ , $\alpha_{n,0}$
   Output: $\alpha_{n,I_n}$, $I_n$
1  Define constants: $\alpha_{min} = 10^{-8}$, flag = 1, $\eta = 2$, $c_2 = 0.9$, $i = 0$
2  $\alpha_{max} = min(\frac{1}{||\boldsymbol{d}_n||_2}, 10^7)$
3  Evaluate $F_n'(0)$, increment $i$ (or use saved gradient from last $F_{n-1}'(\alpha_{n-1,I_{n-1}})$, to
     evaluate $\boldsymbol{g}(\boldsymbol{x}_{n-1} + \alpha_{n-1,I_{n-1}} \cdot \boldsymbol{d}_{n-1})^T \boldsymbol{d}_n$ without incrementing $i$)
4  if $\alpha_{n,0} > \alpha_{max}$ then
5      $\alpha_{n,0} = \alpha_{max}$
6  if $\alpha_{n,0} < \alpha_{min}$ then
7      $\alpha_{n,0} = \alpha_{min}$
8  Evaluate $F_n'(\alpha_{n,0})$, increment $i$
9  Define $tol_{dd} = |c_2 F_n'(0)|$
10 if $F_n'(\alpha_{n,0}) > 0$ and $\alpha_{n,0} < \alpha_{max}$ then
11     flag = 1, decrease step size
12 else if $F_n'(\alpha_{n,0}) < 0$ and $\alpha_{n,0} > \alpha_{min}$ then
13     flag = 2, increase step size
14 else
15     flag= 0
16 while flag > 0 do
17     if flag = 2 then
18         $\alpha_{n,i+1} = \alpha_{n,i} \cdot \eta$
19         Evaluate $F_n'(\alpha_{n,i+1})$
20         if $F_n'(\alpha_{n,i+1}) \geq tol_{dd}$ then
21             flag = 0
22         if $\alpha_{n,i+1} > \frac{\alpha_{max}}{\eta}$ then
23             flag = 0
24     if flag = 1 then
25         $\alpha_{n,i+1} = \frac{\alpha_{n,i}}{\eta}$
26         Evaluate $F_n'(\alpha_{n,i+1})$
27         if $F_n'(\alpha_{n,i+1}) < tol_{dd}$ then
28             flag = 0
29         if $\alpha_{n,i+1} < \alpha_{min} \cdot \eta$ then
30             flag = 0
31 $\alpha_{n,I_n} = \alpha_{n,i+1}$
```

A.3.4 Inexact Line Search: Gradient-Only Line Search with Backtracking (GOLS-Back)

Parameters used for this method are: $\eta = 2$, $c_2 = 0$, $\alpha_{min} = 10^{-8}$ and $\alpha_{max} = 10^7$. $F'_n(\alpha) = \boldsymbol{g}(\boldsymbol{x}_n + \alpha \cdot \boldsymbol{d}_n)^T \boldsymbol{d}_n$.

Algorithm 6: GOLS-Back: Gradient-Only Line Search with Backtracking

```
   Input: $F'_n(\alpha)$, $\boldsymbol{d}_n$ , $\alpha_{n,0}$
   Output: $\alpha_{n,I_n}$, $I_n$
1  Define constants: $\alpha_{min} = 10^{-8}$, flag = 1, $\eta = 2$, $c_2 = 0$, $i = 0$
2  Evaluate $F'_n(0)$, increment $i$ (or use saved gradient from last $F'_{n-1}(\alpha_{n-1,I_{n-1}})$, to
     evaluate $\boldsymbol{g}(\boldsymbol{x}_{n-1} + \alpha_{n-1,I_{n-1}} \cdot \boldsymbol{d}_{n-1})^T \boldsymbol{d}_n$ without incrementing $i$)
3  $\alpha_{max} = min(\frac{1}{||\boldsymbol{d}_n||_2}, 10^7)$
4  $\alpha_{n,0} = \alpha_{max}$
5  Evaluate $F'_n(\alpha_{n,0})$, increment $i$
6  Define $tol_{dd} = |c_2 F'_n(0)|$
7  if $F'_n(\alpha_{n,0}) > 0$ and $\alpha_{n,0} < \alpha_{max}$ then
8      flag = 1, decrease step size
9  if $F'_n(\alpha_{n,0}) < 0$ then
10     flag = 0
11 while flag > 0 do
12     $\alpha_{n,i+1} = \frac{\alpha_{n,i}}{\eta}$
13     Evaluate $F'_n(\alpha_{n,i+1})$
14     if $F'_n(\alpha_{n,i+1}) < 0$ then
15         flag = 0
16     if $\alpha_{n,i+1} < \alpha_{min} \cdot \eta$ then
17         flag = 0
18 $\alpha_{n,I_n} = \alpha_{n,i+1}$
```

A.4 Line Search Adapted optimization algorithms

Here we list the considered popular training algorithms that were adapted to accommodate line searches in Chapter 5. Again, the algorithms are shown using symbols that are agnostic to the mini-batch sub-sampling (MBSS) method used, as both static and dynamic MBSS is possible for these algorithms. Gradient-only line searches adapt seamlessly to both sub-sampling methods.

A.4.1 Line Search Stochastic Gradient Descent (LS-SGD)

Stochastic Gradient Descent (SGD) [Robbins and Monro, 1951] is based on the steepest descent algorithm [Arora, 2011], but uses MBSS loss function approximations, $\boldsymbol{g}(\boldsymbol{x})$ as its search directions. When an *a priori* learning rate schedule has been selected, SGD is equivalent to a subgradient approach [Boyd et al., 2003]. We use line searches to determine its learning rates, called LS-SGD, in Algorithm 7 and coupled it with GOLS-I:

Algorithm 7: LS-SGD: Line Search Stochastic Gradient Descent

```
1  Set $n = 0$ and choose the initial weights $\boldsymbol{x}_0$
2  while stop criterion not met do
3      Compute $\boldsymbol{g}(\boldsymbol{x}_n)$
4      Define the search direction, $\boldsymbol{d}_n = -\boldsymbol{g}(\boldsymbol{x}_n)$
5      Set the step length, $\alpha_{n,I_n}$, using a line search
6      Update $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_{n,I_n} \boldsymbol{x}_n$
```

A.4.2 Line Search Stochastic Gradient Descent with Momentum (LS-SGDM)

The addition of a momentum term to the steepest descent formulation allows for a fraction of the previous update step to be added to the current step [Rumelhart et al., 1988]. This emulates the behaviour of "momentum" in a physical system. The rationale behind this approach is to allow the algorithm to escape local minima with the aid of this "momentum". A consequence thereof is that ascent steps can be taken, in particular if the momentum parameter is large. An outline of the method with a line search is given in Algorithm 8:

Algorithm 8: LS-SGDM: Line Search Stochastic Gradient Descent using Momentum

```
1 Set $n = 0$ and choose initial weights $\boldsymbol{x}_0$, momentum constant $\gamma_m = 0.9$, an initial
    update term $\boldsymbol{c}_0 = \bar{\boldsymbol{0}}$
2 while stop criterion not met do
3     Compute $\boldsymbol{g}(\boldsymbol{x}_n)$
4     Define the descent direction $\boldsymbol{d}_n = -\boldsymbol{g}(\boldsymbol{x}_n)$
5     Set the step length, $\alpha_{n,I_n}$, using a line search
6     Define the update step $\boldsymbol{c}_{n+1} = \alpha_{n,I_n}\boldsymbol{d}_n + \gamma_m\boldsymbol{c}_n$
7     Update $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \boldsymbol{c}_n$
```

A.4.3 Line Search Nesterov Accelerated Gradient Descent (LS-NAG)

Nesterov's Accelerated Gradient Descent algorithm [Nesterov, 1983] can be seen as an extension of the momentum strategy. In this case the gradient vector for the update step is evaluated only once the momentum term has been added to the current solution, as opposed to afterwards as for SGD with momentum. This therefore results in a less naive implementation of the momentum concept. This method as used with a line search is given in Algorithm 9:

Algorithm 9: LS-NAG: Line Search Nesterov Accelerated Gradient Descent

```
1 Set $n = 0$ and choose initial weights $\boldsymbol{x}_0$, an initial update term $\boldsymbol{c}_0 = \bar{\boldsymbol{0}}$, momentum
    constant $\gamma_m = 0.5$
2 while stop criterion not met do
3     Compute $\boldsymbol{g}(\boldsymbol{x}_n + \gamma_m\boldsymbol{c}_n)$
4     Define the search direction $\boldsymbol{d}_n = -\boldsymbol{g}(\boldsymbol{x}_n + \gamma_m\boldsymbol{c}_n)$
5     Define the step length, $\alpha_{n,I_n}$, using a line search
6     Define the update step $\boldsymbol{c}_{n+1} = \alpha_{n,I_n}\boldsymbol{d}_n + \gamma_m\boldsymbol{c}_n$
7     Update $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \boldsymbol{c}_{n+1}$
```

A.4.4 Line Search Adagrad

The Adagrad algorithm [Duchi et al., 2011] performs steepest descent updates with an integrated learning rate scheme independently on each weight. This means, that a learning rate is assigned to every dimension separately, the magnitude of which is determined to be a function of the sum of the current, as well as previous squared gradient magnitudes. The learning rate schedule is constructed such that it biases higher learning rates for dimensions that have a flat slope (low partial derivative magnitudes), and assigns smaller learning rates to dimensions with large slopes (high partial derivative magnitudes). We determine the learning rate for Adagrad using a line search in Algorithm 10.

Algorithm 10: LS-ADAGRAD: Line Search Adagrad

```
1 Set $n = 0$, $\boldsymbol{v}_0 = \bar{\mathbf{0}}$, $\boldsymbol{c}_0 = \bar{\mathbf{0}}$ and choose the initial weights $\boldsymbol{x}_0$
2 while stop criterion not met do
3     Compute $\boldsymbol{g}(\boldsymbol{x}_n)$
4     Define $\boldsymbol{c}_n = -\boldsymbol{g}(\boldsymbol{x}_n)$
5     Calculate $\boldsymbol{v}_{n+1} = \boldsymbol{c}_n \odot \boldsymbol{c}_n + \boldsymbol{v}_n$ with $\odot$ indicating the element-wise multiplication or
        Hadamard product [Reams, 1999]
6     Define the components of the search direction $\boldsymbol{d}_n = (\boldsymbol{v}_{n+1} + \epsilon\bar{\mathbf{1}})^{\circ -\frac{1}{2}} \odot \boldsymbol{c}_n$, with $\bar{\mathbf{1}}$
        indicating a vector with all elements one and the superscript $\circ$ the Hadamard
        power or the power of each element in the vector $(\boldsymbol{v}_{n+1} + \epsilon\bar{\mathbf{1}})$ to $-\frac{1}{2}$.
7     Set the step length, $\alpha_{n,I_n}$, using a line search
8     Update $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_{n,I_n}\boldsymbol{d}_n$
```

A.4.5 Line Search Adadelta

A disadvantage of Adagrad is that the accumulation of all the past gradients in the $\boldsymbol{v}_n$ term causes the $\boldsymbol{v}_n$ term to become large over time, diminishing the search direction.. This causes the overall learning rate to decrease and can cause slow progress in later stages of training. Adadelta [Zeiler, 2012] is an extension of Adagrad, which makes use of an exponentially decaying average for $\boldsymbol{v}_n$, such that a reasonable learning rate remains throughout training. It also implements and update magnitude rule in the form of an exponentially decaying average of the previous updates, $\boldsymbol{m}_n$. We add a line search to this method in Algorithm 11:

Algorithm 11: LS-ADADELTA: Line Search Adadelta

```
1 Set $n = 0$, $\boldsymbol{v}_0 = \bar{\mathbf{0}}$, $\boldsymbol{d}_0 = \bar{\mathbf{1}}$ and choose the initial weights $\boldsymbol{x}_0$ and $\beta = 0.9$
2 while stop criterion not met do
3     Compute $\boldsymbol{g}(\boldsymbol{x}_n)$
4     Define $\boldsymbol{c}_n = -\boldsymbol{g}(\boldsymbol{x}_n)$
5     Calculate $\boldsymbol{v}_{n+1} = (\beta - 1)\boldsymbol{c}_n \odot \boldsymbol{c}_n + \beta\boldsymbol{v}_n$
6     Calculate $\boldsymbol{m}_{n+1} = (\beta - 1)\boldsymbol{d}_n \odot \boldsymbol{d}_n + \beta\boldsymbol{m}_n$
7     Define the components of the search direction
        $\boldsymbol{d}_{n+1} = (\boldsymbol{m}_{n+1} + \epsilon\bar{\mathbf{1}})^{\circ\frac{1}{2}} \odot (\boldsymbol{v}_{n+1} + \epsilon\bar{\mathbf{1}})^{\circ -\frac{1}{2}} \odot \boldsymbol{c}_n$
8     Determine $\alpha_{n,I_n}$, using a line search
9     Define $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_{n,I_n}\boldsymbol{d}_{n+1}$
```

A.4.6 Line Search Adam

Adaptive Moment Estimation (Adam) [Kingma and Ba, 2015] also makes use of different learning rates for independent components of $\boldsymbol{x}_n$. These learning rates are a function of exponentially decaying past averages. In this case these are obtained from the first moment (the mean) $\boldsymbol{m}_n$ and the second moment (the centred variance) $\boldsymbol{v}_n$ of the past gradients. Due to the initial values for both these variables being chosen to be $\mathbf{0}$, the initial learning rates tend to be too small, resulting in slow training in the beginning. To account for this, the respective bias-corrected estimates are used, $\hat{\boldsymbol{m}}_n$ and $\hat{\boldsymbol{v}}_n$. We determine its learning rates using a line search in Algorithm 12:

Algorithm 12: LS-Adam: Line Search Adam

```
1  Set $n = 0$, $\boldsymbol{m}_0 = \bar{\mathbf{0}}$, $\boldsymbol{v}_0 = \bar{\mathbf{0}}$, and choose the initial weights $\boldsymbol{x}_0$, $\beta_1 = 0.9$ ($\beta_1 = 0$) and
   $\beta_2 = 0.999$
2  while stop criterion not met do
3      Compute $\boldsymbol{g}(\boldsymbol{x}_n)$
4      Define $\boldsymbol{c}_n = \boldsymbol{g}(\boldsymbol{x}_n)$
5      Define $\boldsymbol{m}_{n+1} = \beta_1 \boldsymbol{m}_n + (1 - \beta_1)(\boldsymbol{c}_n)$
6      Define $\boldsymbol{v}_{n+1} = \beta_2 \boldsymbol{v}_n + (1 - \beta_2)(\boldsymbol{c}_n \odot \boldsymbol{c}_n)$
7      Calculate $\hat{\boldsymbol{m}}_{n+1} = \frac{\boldsymbol{m}_{n+1}}{1-\beta_1}$
8      Calculate $\hat{\boldsymbol{v}}_{n+1} = \frac{\boldsymbol{v}_{n+1}}{1-\beta_2}$
9      Define the components of the search direction $\boldsymbol{d}_n = (\hat{\boldsymbol{v}}_{n+1}^{\circ\frac{1}{2}} + \epsilon\bar{\mathbf{1}})^{\circ-1} \odot \hat{\boldsymbol{m}}_{n+1}$
10     Set the step length, $\alpha_{n,I_n}$, using a line search
11     Update $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_{n,I_n}\boldsymbol{d}_n$
```

A.5 Heuristics for determining the number of hidden units

The number of hidden units in a single hidden layer feedforward network, H, was chosen to be the smaller of two heuristics, H_1 and H_2 given as:

$$H_1 = \frac{\frac{M}{C_r} - K}{D + K + 1}, \tag{A.4}$$

and

$$H_2 = D - 1. \tag{A.5}$$

And therefore,

$$H = min(H_1, H_2), \tag{A.6}$$

where D is the number of input features of a given dataset, M is the total number of observations, C_r is a regression constant and H_1 is rounded down to the nearest integer value. The regression constant C_r determines how rigid the model is, with $C_r > 1$ resulting in less parameters relative to the degrees of freedom in the data, and $C_r < 1$ resulting in more parameters than data points in the training dataset. In our investigation this constant was set to $C_r = 1.5$, which ensures that the model regresses through the data. Arguably, pruning [Reed, 1993] is an alternative approach, but our heuristic approach is sufficient for the purposes of this investigation with the benefit of simplicity.

Bibliography

A. Agarwal, S. Negahban, and M. J. Wainwright. Noisy Matrix Decomposition via Convex Relaxation: Optimal Rates in High Dimensions. *Annals of Statistics*, 40(2):1171–1197, 2012. ISSN 00905364. doi: 10.1214/12-AOS1000.

M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. de Freitas. Learning to Learn by Gradient Descent by Gradient Descent. In *NIPS 2016*, pages 1–17, 2016. ISBN 1011500801515. doi: 10.1007/s10115-008-0151-5.

D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz. Human Activity Recognition on Smartphones using a Multiclass Hardware-friendly Support Vector Machine. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 7657 LNCS, pages 216–223, 2012. ISBN 9783642353949. doi: 10.1007/978-3-642-35395-6_30.

M. Anitescu. Degenerate Nonlinear Programming with a Quadratic Growth Condition. *SIAM Journal on Optimization*, 10(4):1116–1135, January 2000. ISSN 1052-6234. doi: 10.1137/S1052623499359178.

W.B. Aribi, H. Ammar, and M.B. Alaya. Stochastic global optimization using tangent minorants for lipschitz functions. *Journal of Computational and Applied Mathematics*, in print:1–15, 2019.

J. Arora. *Introduction to Optimum Design, Third Edition.* Academic Press Inc, 2011. ISBN 0123813751.

L. Balles and P. Hennig. Dissecting Adam: The Sign, Magnitude and Variance of Stochastic Gradients. *arXiv:1705.07774v2 [cs.LG]*, 1:1–17, 2018.

I. Bello, B. Zoph, V. Vasudevan, and Q. V. Le. Neural Optimizer Search with Reinforcement Learning. *arXiv:1709.07417*, 2017. ISSN 1938-7228. doi: 10.1016/j.knosys.2015.01.010.

Y. Bengio. *Learning Deep Architectures for AI.* Foundations and Trends in Machine Learning. Now Publishers Inc, 2009. ISBN 1601982941,9781601982940.

E. Bergou, Y. Diouane, V. Kungurtsev, and C. W. Royer. A Subsampling Line-Search Method with Second-Order Results. *arXiv:1810.07211*, pages 1–29, October 2018.

J. Bergstra and Y. Bengio. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13(February):281–305, 2012. ISSN 1532-4435. doi: 10.1162/153244303322533223.

J. Bergstra, G. Desjardins, P. Lamblin, and Y. Bengio. Quadratic Polynomials Learn Better Image Features. In *Technical Report 1337, IT Department and operations research, University of Montreal*, pages 1–11, 2009.

J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for Hyper-Parameter Optimization. In *NIPS 2011*, pages 2546–2554, 2011. ISBN 9781618395993. doi: 2012arXiv1206.2944S.

D. P. Bertsekas and Massachusetts Institute of Technology. *Convex Optimization Algorithms, First Edition*. Athena Scientific, 2015. ISBN 1886529280.

B. Bischl, G. Casalicchio, M. Feurer, F. Hutter, M. Lang, R. G. Mantovani, J. N. Van Rijn, and J. Vanschoren. OpenML Benchmarking Suites and the OpenML100. *arXiv:1708.0373*, pages 1–6, 2017.

C. M. Bishop. *Pattern Recognition and Machine Learning, First Edition*. Springer, 2006. ISBN 9781493938438.

R. Bollapragada, R. Byrd, and J. Nocedal. Adaptive Sampling Strategies for Stochastic Optimization. *arXiv:1710.11258*, pages 1–32, 2017.

L. Bottou. Large-Scale Machine Learning with Stochastic Gradient Descent. In *COMPSTAT 2010, Keynote, Invited and Contributed Papers*, volume 19, pages 177–186, 2010. ISBN 9783790826036. doi: 10.1007/978-3-7908-2604-3-16.

S. Boyd and J. Park. Subgradient Methods. *Lecture notes of EE365b, Stanford University*, 1 (May):1–39, 2014.

S. Boyd, L. Xiao, and A. Mutapcic. Subgradient Methods. In *lecture notes of EE392o, Stanford University*, volume 1, pages 1–21, 2003.

R. H. Byrd, G. M. Chin, W. Neveitt, and J. Nocedal. On the Use of Stochastic Hessian Information in Optimization Methods for Machine Learning. *SIAM Journal on Optimization*, 21 (3):977–995, 2011. ISSN 1052-6234. doi: 10.1137/10079923x.

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 2012. ISSN 0025-5610. doi: 10.1007/s10107-012-0572-5.

Y. Chae and D. N. Wilke. Empirical study towards understanding line search approximations for training neural networks. *arXiv:1909.06893 [stat.ML]*, pages 1–30, September 2019.

T. Chen, Y. Sun, Y. Shi, and L. Hong. On Sampling Strategies for Neural Network-based Collaborative Filtering. *arXiv:1706.07881 [cs.LG]*, pages 1–14, 2017.

A. Chevallier and M. Jakubowska. Application of chemometric methods in analysis of environmental data. *Analit*, 6:84–93, 2018.

A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun. The Loss Surfaces of Multilayer Networks. In *AISTATS 2015*, volume 38, pages 192–204, 2015. ISBN 1412.0233.

D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUS). In *ICLR 2016*, pages 1–14, 2016. ISBN 9780996452700. doi: 10.23919/ICIF.2017.8009734.

D. Csiba and P. Richtárik. Importance Sampling for Minibatches. *Journal of Machine Learning Research*, 19:1–21, 2018.

C. Darken and J. E. Moody. Note on Learning Rate Schedules for Stochastic Optimization. In *NIPS 1990*, volume 3, pages 832–838, 1990. ISBN 1-55860-100-7.

Y. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the Saddle Point Problem in High-Dimensional Non-Convex Optimization. In *ICLR 2014*, pages 1–9, 2014. ISBN 1406.2572.

C. Davis. The Norm of the Schur Product Operation. *Numerische Mathematik*, 4(1):343–344, December 1962. ISSN 0029-599X. doi: 10.1007/BF01386329.

O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 13:165–202, December 2012.

M. Denkowski and G. Neubig. Stronger Baselines for Trustable Results in Neural Machine Translation. *arXiv:1706.09733*, pages 1–10, 2017.

J. Duchi, E. Hazan, and Y. Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12(July):2121–2159, 2011. ISSN ISSN 1533-7928.

A. P. Engelbrecht. *Fundamentals of Computational Swarm Intelligence, First Edition.* Wiley, 2005. ISBN 0470091916. URL `http://si.cs.up.ac.za/`.

R. A. Fisher. The use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics*, 7(2):179–188, September 1936. ISSN 20501420. doi: 10.1111/j.1469-1809.1936.tb02137.x.

C. A. Floudas and P. M. Pardalos. *Encyclopedia of Optimization, Second Edition.* Springer, 2009. ISBN 978-0-387-74758-3.

M. P. Friedlander and M. Schmidt. Hybrid Deterministic-Stochastic Methods for Data Fitting. *arXiv:1104.2373 [cs.LG]*, pages 1–26, 2011. doi: 10.1137/110830629.

M. Gaviano and D. Lera. A global minimization algorithm for lipschitz functions. *Optimization Letters*, 2:1–13, 2008.

X. Glorot and Y. Bengio. Understanding the Difficulty of Training Deep Feedforward Neural Networks. *arXiv:1308.0850*, pages 1–8, 2010. ISSN 15324435. doi: 10.1109/IJCNN.1993.716981.

X. Glorot and A. Bordes. Deep Sparse Rectifier Neural Networks. In *Proceedings of Machine Learning Research*, volume 15, pages 315–323, 2011. ISBN 1532-4435. doi: 10.1.1.208.6449.

P. Gong and J. Ye. Linear Convergence of Variance-Reduced Stochastic Gradient without Strong Convexity. *arXiv:1406.1102*, June 2014.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

I. J. Goodfellow, O. Vinyals, and A. M. Saxe. Qualitatively Characterizing Neural Network Optimization Problems. In *ICLR 2015*, pages 1–11, 2015.

J. Han and C. Morag. The Influence of the Sigmoid Function Parameters on the Speed of Backpropagation Learning. In José Mira and Francisco Sandoval, editors, *Natural to Artificial Neural Computation. Lecture Notes in Computer Science*, volume 930. Springer, 1995. ISBN 978-3-540-59497-0.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv:1502.01852 [cs.CV]*, pages 1–11, February 2015.

K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *IEEE 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) - Las Vegas, NV, USA (2016.6.27-2016.6.30)*, 2016. ISBN 978-1-4673-8851-1. doi: 10.1109/CVPR.2016.90.

P. Hennig. Fast probabilistic optimization from noisy gradients. In *In Proceedings of the 30th International Conference on Machine Learning*, volume 28, pages 62–70, 2013.

S. Hochreiter and P. Frasconi. Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies. *A Field Guide to Dynamical Recurrent Networks*, 2009. ISSN 1098-6596. doi: 10.1109/9780470544037.ch14.

Q. Huang, K. Zhou, S. You, and U. Neumann. Learning to Prune Filters in Convolutional Neural Networks. In *Proceedings - 2018 IEEE Winter Conference on Applications of Computer Vision, WACV 2018*, volume 2018-January, pages 709–718, 2018. ISBN 9781538648865. doi: 10.1109/WACV.2018.00083.

D. J. Im, M. Tao, and K. Branson. An Empirical Analysis of the Optimization of Deep Network Loss Surfaces. *arXiv:1612.04010*, pages 1–21, 2016.

S. Ioffe. Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models. *arXiv:1702.03275v2*, 2017.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv:1502.03167 [cs.LG]*, pages 1–9, February 2015.

M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu. Population Based Training of Neural Networks. *arXiv:1711.09846*, pages 1–13, 2017.

B. Johnson, R. Tateishi, and Z. Xie. Using Geographically Weighted Variables for Image Classification. *Remote Sensing Letters*, 3(6):491–499, November 2012. ISSN 2150-704X. doi: 10.1080/01431161.2011.629637.

B. A. Johnson, R. Tateishi, and N. T. Hoan. A Hybrid Pansharpening Approach and Multiscale Object-Based Image Analysis for Mapping Diseased Pine and Oak Trees. *International Journal of Remote Sensing*, 34(20):6969–6982, October 2013. ISSN 0143-1161. doi: 10.1080/01431161.2013.810825.

D. Kafka and D. N. Wilke. Gradient-Only Line Searches: An Alternative to Probabilistic Line Searches. *arXiv:1903.09383 [stat.ML]*, pages 1–25, March 2019.

D. Kafka and D. N. Wilke. Visual Interpretation of the Robustness of Non-Negative Associative Gradient Projection Points over Function Minimizers in Mini-Batch Sampled Loss Functions. *arXiv:1903.08552 [stat.ML]*, pages 1–32, March 2019a.

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 2019b.

H. Karimi, J. Nutini, and M. Schmidt. Linear Convergence of Gradient and Proximal-Gradient Methods Under the Polyak-Łojasiewicz Condition. In *ECML PKDD: Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, volume 9851, pages 795–811. Springer, Cham, September 2016. doi: 10.1007/978-3-319-46128-1_50.

B. Karlik. Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks. *International Journal of Artificial Intelligence and Expert Systems*, 1(4):111–122, 2015.

R.L. Kashyap, C.C. Blaydon, and K.S. Fu. 9 Stochastic Approximation. In J.M. Mendel and K.S. Fu, editors, *Adaptive, Learning and Pattern Recognition Systems*, volume 66 of *Mathematics in Science and Engineering*, pages 329 – 355. Elsevier, 1970. doi: https://doi.org/10.1016/S0076-5392(08)60499-3.

K. Kawaguchi. Deep Learning without Poor Local Minima. In *NIPS 2016*, 2016. ISBN 9781627480031. doi: 10.1177/0278364910371999.

D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *ICLR 2015*, pages 1–15, 2015. ISBN 9781450300728. doi: 10.1145/1830483.1830503.

D. P Kingma and M. Welling. Auto-Encoding Variational Bayes. *arXiv:1312.6114v10*, pages 1–14, 2013. ISSN 1312.6114v10. doi: 10.1051/0004-6361/201527329.

S. Kirkpatrick, C. D. Gelatt, and P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, May 1983.

R. Kleinberg, Y. Li, and Y. Yuan. An Alternative View: When Does SGD Escape Local Minima? In *35 th International Conference on Machine Learning (PMLR 80)*, pages 1–10, 2018.

A. Krizhevsky and G. E. Hinton. Learning Multiple Layers of Features from Tiny Images. *University of Toronto*, 2009. URL `https://www.cs.toronto.edu/~kriz/cifar.html`.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS 2012*, pages 1–9, 2012. ISBN 9781420010749. doi: 10.1201/9781420010749.

V. Kungurtsev and T Pevny. Algorithms for solving optimization problems arising from deep neural net models: smooth problems. *arXiv:1807.00172 [math.OC]*, pages 1–5, June 2018.

D.E. Kvasov, D. Yarsolav, and D. Sergeyev. Lipschitz gradients for global optimization in a one-point-based partitioning scheme. *Journal of Computational and Applied Mathematics*, 236:4042–4054, 2012.

A. Laudani, G. M. Lozito, F. R. Fulginei, and A. Salvini. On Training Efficiency and Computational Costs of a Feed Forward Neural Network: A Review. *Computational Intelligence and Neuroscience*, 2015:1–13, 2015. ISSN 16875273. doi: 10.1155/2015/818243.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. Visualizing the Loss Landscape of Neural Nets. *arXiv:1712.09913*, pages 1–21, 2017.

M. Li, T. Zhang, Y. Chen, and A. J. Smola. Efficient Mini-batch Training for Stochastic Optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1–10, 2014. ISBN 9781450329569. doi: 10.1145/2623330.2623612.

S. Liang, R. Sun, J. D. Lee, and R. Srikant. Adding One Neuron Can Eliminate All Bad Local Minima. volume 32, 2018. doi: 10.1111/ajt.12036.

J. Liu, S. J. Wright, C. Ré, V. Bittorf, and S. Sridhar. An Asynchronous Parallel Stochastic Coordinate Descent Algorithm. *Journal of Machine Learning Research*, 16:285–322, 2015.

K. Liu. 95.16% on CIFAR10 with PyTorch. `https://github.com/kuangliu/pytorch-cifar`, 2018. Accessed: 2018-09-30.

I. Loshchilov and F. Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts. *arXiv:1608.03983*, pages 1–16, 2016. ISSN 15826163. doi: 10.1002/fut.

D. D. Lucas, R. Klein, J. Tannahill, D. Ivanova, S. Brandon, D. Domyancic, and Y. Zhang. Failure Analysis of Parameter-Induced Simulation Crashes in Climate Models. *Geoscientific Model Development*, 6(4):1157–1171, August 2013. ISSN 1991-9603. doi: 10.5194/gmd-6-1157-2013.

Z.-Q. Luo and P. Tseng. Error Bounds and Convergence Analysis of Feasible Descent Methods: A General Approach. *Annals of Operations Research*, 46-47(1):157–178, March 1993. ISSN 0254-5330. doi: 10.1007/BF02096261.

A. M. Lyapunov. The General Problem of the Stability of Motion. *International Journal of Control*, 55(3):531–534, 1992. doi: 10.1080/00207179208934253.

A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier Nonlinearities improve Neural Network Acoustic Models. In *ICML 2013*, volume 28, page 6, 2013. ISBN 0162-4962. doi: 10.1016/0010-0277(84)90022-2.

M. Mahsereci and P. Hennig. Probabilistic Line Searches for Stochastic Optimization. *Journal of Machine Learning Research*, 18:1–59, November 2017.

K. Mansouri, T. Ringsted, D. Ballabio, R. Todeschini, and V. Consonni. Quantitative Structure–Activity Relationship Models for Ready Biodegradability of Chemicals. *Journal of Chemical Information and Modeling*, 53(4):867–878, April 2013. ISSN 1549-9596. doi: 10.1021/ci4000213.

J. Martens. Deep Learning via Hessian-free Optimization. In *ICML 2010*, pages 1–6, 2010. ISBN 2090-1291 (Electronic)\r2090-1283 (Linking). doi: 10.1155/2011/176802.

T. Marwala. Bayesian Training of Neural Networks using Genetic Programming. *Pattern Recognition Letters*, 28(12):1452–1458, September 2007. ISSN 0167-8655. doi: 10.1016/J.PATREC.2007.03.004.

D. Masters and C. Luschi. Revisiting Small Batch Training for Deep Neural Networks. *arXiv:1804.07612*, pages 1–18, 2018.

Mathworks. Matlab. `https://www.mathworks.com/products/matlab.html`, 2015. Version: R2015b.

D. J. Montana and L. Davis. Training feedforward neural networks using genetic algorithms, 1989.

M. Moreira and E. Fiesler. Neural Networks with Adaptive Learning Rate and Momentum Terms. *Technique Report 95*, 4:1–29, 1995.

P. M. Morse and H. Feshbach. *Methods of Theoretical Physics. Parts I and II*, volume 1. McGraw-Hill Book Co., New York, 1953.

M. C. Mukkamala and M. Hein. Variants of RMSProp and Adagrad with Logarithmic Regret Bounds. In *International Conference on Machine Learning*, pages 1–16, 2017. ISBN 9781510855144.

M. Mutschler and A. Zell. Parabolic Approximation Line Search: An efficient and effective line search approach for DNNs. *arXiv:1903.11991[cs.LG]*, pages 1–16, Mar 2019.

W. J. Nash, T. L. Sellers, S. R. Talbot, A. J. Cawthorn, and W. B. Ford. The Population Biology of Abalone (_Haliotis_ species) in Tasmania. I. Blacklip Abalone (_H. rubra_) from the North Coast and Islands of Bass Strait. Technical report, Sea Fisheries Division, 1994.

Y. Nesterov. A Method of Solving a Convex Programming Problem with Convergence Rate $O(1/k^2)$. *Soviet Mathematics Doklady*, 27:372–376, 1983.

Y. Nesterov. Primal-Dual Subgradient Methods for Convex Problems. *Math. Program., Ser. B*, 120:221–259, 2009. doi: 10.1007/s10107-007-0149-x.

Q. Nguyen, M. C. Mukkamala, and M. Hein. On The Loss Landscape of a Class of Deep Neural Networks with no Bad Local Valleys. *arXiv:1809.10749*, pages 1–20, 2018.

P. Y. Nie. An SQP Approach with Line Search for a System of Nonlinear Equations. *Mathematical and Computer Modelling*, 43(3-4):368–373, 2006. ISSN 08957177. doi: 10.1016/j.mcm.2005.10.007.

J. Nocedal. Updating Quasi-Newton Matrices with Limited Storage. *Mathematics of Computation*, 35:773–782, 1980. doi: 10.1090/S0025-5718-1980-0572855-7.

J. Nocedal and S. J. Wright. *Numerical Optimization*, volume 2. 1999. ISBN 9780387303031. doi: 10.1103/PhysRev.83.134.

F. Orabona and T. Tommasi. Training Deep Networks without Learning Rates Through Coin Betting. *arXiv:1705.07795*, pages 1–14, 2017. ISSN 10495258.

C. Paquette and K. Scheinberg. A Stochastic Line Search Method with Convergence Rate Analysis. *arXiv:1807.07994*, pages 1–26, July 2018.

F. Paschke, C. Bayer, M. Bator, U. Mönks, A. Dicks, O. Enge-Rosenblatt, and V. Lohweg. Sensorlose Zustandsüberwachung an Synchronmotoren. In *Conference: 23. Workshop Computational Intelligence (VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik (GMA))*, Dortmund, 2013.

S. Pouyanfar and S. C. Chen. T-LRA: Trend-Based Learning Rate Annealing for Deep Neural Networks. In *BigMM 2017*, pages 50–57, 2017. ISBN 9781509065493. doi: 10.1109/BigMM.2017.36.

L. Prechelt. PROBEN1 - a set of neural network benchmark problems and benchmarking rules (Technical Report 21-94). Technical report, Universität Karlsruhe, 1994.

pytorch.org. PyTorch. `https://pytorch.org/`, 2019. Version: 1.0.

P. M. Radiuk. Impact of Training Set Batch Size on the Performance of Convolutional Neural Networks for Diverse Datasets. *Information Technology and Management Science*, 20(1): 20–24, 2017. ISSN 2255-9094. doi: 10.1515/itms-2017-0003.

R. Reams. Hadamard Inverses, Square Roots and Products of almost Semidefinite Matrices. *Linear Algebra and its Applications*, 288:35–43, 1999. ISSN 00243795. doi: 10.1016/S0024-3795(98)10162-3.

R. Reed. Pruning Algorithms - a Survey. In *IEEE Transactions on Neural Networks*, volume 4, pages 740–747, 1993. ISBN 10459227 (ISSN). doi: 10.1109/72.248452.

H. Robbins and S. Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400–407, September 1951. ISSN 0003-4851. doi: 10.1214/aoms/1177729586.

S. Ruder. An Overview of Gradient Descent Optimization Algorithms. *arXiv:1609.04747v2 [cs.LG]*, pages 1–14, 2016. ISSN 0006341X. doi: 10.1111/j.0006-341X.1999.00591.x.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Neurocomputing: Foundations of Research*. MIT Press, Cambridge, MA, USA, 1988. ISBN 0-262-01097-6.

Tim Salimans and Diederik P. Kingma. Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks. In *NIPS 2016*, pages 1–11, 2016. ISBN 9781450300728.

A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact Solutions to the Nonlinear Dynamics of Learning in Deep Linear Neural Networks. *CoRR*, abs/1312.6:1–22, 2013.

N. N. Schraudolph. Local Gain Adaptation in Stochastic Gradient Descent. *9th International Conference on Artificial Neural Networks: ICANN '99*, 1999:569–574, 1999. ISSN 0537-9989. doi: 10.1049/cp:19991170.

N. N. Schraudolph and T. Graepel. Combining conjugate direction methods with stochastic approximation of gradients. *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics, AISTATS 2003*, pages 2–7, 2003.

N. N. Schraudolph, J. Yu, and S. Günter. A Stochastic Quasi-Newton Method for Online Convex Optimization. *International Conference on Artificial Intelligence and Statistics*, pages 436—-443, 2007. ISSN 15324435. doi: 10.1137/140954362.

A. Senior, G. Heigold, M.-A. Ranzato, and K. Yang. An Empirical Study of Learning Rates in Deep Neural Networks for Speech Recognition. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013*, pages 6724–6728, 2013. ISSN 1520-6149. doi: 10.1109/ICASSP.2013.6638963.

A. Serwa. Studying the Effect of Activation Function on Classification Accuracy Using Deep Artificial Neural Networks. *Journal of Remote Sensing & GIS*, 06(03):6–11, 2017. doi: 10.4172/2469-4134.1000203.

N. Z. Shor. *Minimization Methods for Non-Differentiable Functions, First Edition.* Springer, Berlin Heidelberg, 1985a. ISBN 9783642821202.

N. Z. Shor. The subgradient method. In *Minimization Methods for Non-Differentiable Functions*, pages 22–47. Springer, Berlin Heidelberg, 1985b.

U. Simsekli, L. Sagun, and M. Gurbuzbalaban. A Tail-Index Analysis of Stochastic Gradient Noise in Deep Neural Networks. *arXiv:1901.06053*, pages 1–14, 2019.

L. N. Smith. Cyclical Learning Rates for Training Neural Networks. *arXiv:1506.01186*, 2015. doi: 10.1109/WACV.2017.58.

S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le. Don't Decay the Learning Rate, Increase the Batch Size. *arXiv:1711.00489*, 2017.

J. Snoek, H. Larochelle, and R.P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In *NIPS*, pages 1–9, 2012. ISBN 9781627480031. doi: 2012arXiv1206.2944S.

J. A. Snyman and D. N. Wilke. *Practical Mathematical Optimization*, volume 133 of *Springer Optimization and Its Applications*. Springer International Publishing, Cham, 2018. ISBN 978-3-319-77585-2. doi: 10.1007/978-3-319-77586-9.

W. N. Street, W. H. Wolberg, and O. L. Mangasarian. Nuclear Feature Extraction For Breast Tumor Diagnosis. *International Symposium on Electronic Imaging: Science and Technology*, 1905:861–870, 1993.

R.G. Strongin, D. Yaroslav, and D. Sergeyev. *Global Optimization with Non-Convex Constraints: Sequential and Parallel Algorithms*, volume 1. Dordrecht ; Boston : Kluwer Academic Publishers, 2000.

tensorflow.org. TensorFlow. `https://www.tensorflow.org/`, 2019. Version: 2.0.

F. Tong and X. Liu. Samples Selection for Artificial Neural Network Training in Preliminary Structural Design. *Tsinghua Science & Technology*, 10(2):233–239, April 2005. ISSN 1007-0214. doi: 10.1016/S1007-0214(05)70060-2.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention Is All You Need. In *NIPS 2017*, pages 1–15, 2017. ISBN 9781577357384. doi: 10.1017/S0140525X16001837.

M. Vurkaç. Clave-Direction Analysis: A New Arena for Educational and Creative Applications of Music Technology. *Journal of Music, Technology and Education*, 4(1):27–46, August 2011. ISSN 17527066. doi: 10.1386/jmte.4.1.27_1.

A. Wächter and L. Biegler. Line Search Filter Methods for Nonlinear Programming: Local Convergence. *SIAM Journal on Optimization*, 16(1):32–48, 2005. doi: 10.1137/S1052623403426544.

S. Wenkel. TPOT for sensorless drive diagnosis. `https://www.simonwenkel.com/2018/10/16/TPOT-for-sensorless-drive-diagnosis.html`, 2018. Accessed: 2018-09-10.

P. J. Werbos. Applications of Advances in Nonlinear Sensitivity Analysis. In R. F. Drenick and F. Kozin, editors, *System Modeling and Optimization. Lecture Notes in Control and Information Sciences*, volume 38. Springer, Berlin, Heidelberg, 1982. ISBN 3-540-11691-5.

P. J. Werbos. *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting.* Wiley-Interscience, New York, NY, USA, 1994. ISBN 0-471-59897-6.

D. N. Wilke. Modified Subgradient Methods for Remeshing Based Structural Shape Optimization. In *Thirteenth International Conference on Civil, Structural and Environmental Engineering Computing*, volume 13, pages 1–8, 2011. doi: 10.4203/ccp.96.99.

D. N. Wilke. Structural Shape Optimization using Shor's R-algorithm. In *Third International Conference on Engineering Optimization*, 2012. ISBN 978-85-76503-43-9.

D. N. Wilke, S. Kok, J. A. Snyman, and A. A. Groenwold. Gradient-Only Approaches to Avoid Spurious Local Minima in Unconstrained Optimization. *Optimization and Engineering*, 14 (2):275–304, June 2013. ISSN 1389-4420. doi: 10.1007/s11081-011-9178-7.

A. Wills and T. Schön. Stochastic quasi-Newton with adaptive step lengths for large-scale problems. *arXiv:1802.04310 [stat.ML]*, pages 1–17, February 2018.

A. Wills and T. Schön. Stochastic quasi-Newton with line-search regularization. *arXiv:1909.01238 [eess.SY]*, pages 1–15, September 2019.

D. R. Wilson and T. R. Martinez. The General Inefficiency of Batch Training for Gradient Descent Learning. *Neural Networks*, 16(10):1429–1451, 2003. ISSN 08936080. doi: 10.1016/S0893-6080(03)00138-2.

X. Wu, R. Ward, and L. Bottou. WNGrad: Learn the Learning Rate in Gradient Descent. *arXiv:1803.02865*, pages 1–16, 2018.

B. Xu, R. Huang, and M. Li. Revise Saturated Activation Functions. *arXiv:1602.05980[cs.LG]*, pages 1–7, February 2016.

C. Xu, T. Qin, G. Wang, and T.-Y. Liu. Reinforcement Learning for Learning Rate Control. *arXiv:1705.11159*, pages 1–7, 2017.

I.-C. Yeh and C.-H. Lien. The Comparisons of Data Mining Techniques for the Predictive Accuracy of Probability of Default of Credit Card Clients. *Expert Systems with Applications*, 36(2):2473–2480, March 2009. ISSN 0957-4174. doi: 10.1016/J.ESWA.2007.12.020.

X.-H. Yu. Can Backpropagation Error Surface not have Local Minima. *IEEE Transactions on Neural Networks*, 3, November 1992. doi: 10.1109/72.165604.

M. D. Zeiler. ADADELTA: An Adaptive Learning Rate Method. *arXiv:1212.5701*, pages 1–6, 2012. ISSN 09252312. doi: http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503.

C. Zhang, C. Öztireli, S. Mandt, and G. Salvi. Active Mini-Batch Sampling using Repulsive Point Processes. *ArXiv:1804.02772*, April 2018.

H. Zhang and W. Yin. Gradient Methods for Convex Minimization: Better Rates Under Weaker Conditions. *ArXiv e-prints*, March 2013.

J. Zhang and I. Mitliagkas. YellowFin and the Art of Momentum Tuning. *arXiv:1706.03471*, pages 1–27, 2017.

S. Zheng and J. T. Kwok. Follow the Moving Leader in Deep Learning. In *ICML 2017*, volume 70, pages 4110–4119, 2017. ISBN 9781510855144.

M. Zięba, S. K. Tomczak, and J. M. Tomczak. Ensemble Boosted Trees with Synthetic Features Generation in Application to Bankruptcy Prediction. *Expert Systems with Applications*, 58: 93–101, October 2016. ISSN 0957-4174. doi: 10.1016/J.ESWA.2016.04.001.

X. Zuo and S. Chintala. Basic vae example. `https://github.com/pytorch/examples/tree/master/vae`, 2018. Accessed: 2018-06-07.