Optimisation for Efficient Deep Learning

Alasdair James Paren

St Anne's College

A thesis presented for the degree of

Doctor of Philosophy

Department of Engineering Science

University of Oxford

Trinity 2022

Acknowledgements

I would first like to thank Toshiba Research Laboratory Cambridge for partially funding my doctorate, which would not have been possible without their financial support.

I would next like to acknowledge my dedicated supervisors Pawan and Rudra for their patience throughout my time at the University of Oxford. Rudra provided me with great advice and encouragement, on both technical and non-technical matters. I'd like to thank Pawan for his engaging teaching, very thorough proofreading and attention to detail. I am especially grateful for Pawan's decision to keep supervising me even after leaving his role at Oxford.

I would also like to thank my lab mates Rudy, Alban, Prateek, Jodie, Florian and Alessandro; it was a real pleasure to work alongside you and discuss all sorts of interesting topics and ideas with you. I'm sad COVID stole further opportunities to interact with all of you in person.

I would like to let my parents Julian and Mary know how much I appreciate their support. This including their investment in my education from an early age and their continual perspective and kind words of encouragement.

Finally, I would like to extend a special thank you to my partner, Maria. I'm not going to say "I could not have done it without you", but it certainly would have taken me even longer and I would not have enjoyed it as much. Thank you for the walks, proof reading and cooking around deadlines. Thank you for your continued love and support, it has been a privilege to spend this time with you!

Abstract

Over the past 10 years there has been a huge advance in the performance power of deep neural networks on many supervised learning tasks. Over this period these models have redefined the state of the art numerous times on many classic machine vision and natural language processing benchmarks. Deep neural networks have also found their way into many real-world applications including chat bots, art generation, voice activated virtual assistants, surveillance, and medical diagnosis systems. Much of the improved performance of these models can be attributed to an increase in scale, which in turn has raised computation and energy costs.

In this thesis we detail approaches of how to reduce the cost of deploying deep neural networks in various settings. We first focus on training efficiency, and to that end we present two optimisation techniques that produce high accuracy models without extensive tuning. These optimisers only have a single fixed maximal step size hyperparameter to cross-validate and we demonstrate that they outperform other comparable methods in a wide range of settings. These approaches do not require the onerous process of finding a good learning rate schedule, which often requires training many versions of the same network, hence they reduce the computation needed. The first of these optimisers is a novel bundle method designed for the interpolation setting. The second demonstrates the effectiveness of a Polyak-like step size in combination with an online estimate of the optimal loss value in the non-interpolating setting.

Next, we turn our attention to training efficient binary networks with both binary parameters and activations. With the right implementation, fully binary networks are highly efficient at inference time, as they can replace the majority of operations with cheaper bit-wise alternatives. This makes them well suited for lightweight or embedded applications. Due to the discrete nature of these models conventional training approaches are not viable. We present a simple and effective alternative to the existing optimisation techniques for these models.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

During the past decade deep neural networks have been extremely successful when applied to many supervised learning tasks (LeCun et al., 2015). The areas of machine vision and natural language processing have seen significant progress when adopting neural network based approaches (Krizhevsky et al., 2012; Devlin et al., 2019). Since these successes, the performance of deep neural networks has steadily improved mainly due to increased model sizes, data set sizes and refined training techniques. Alongside progress in research settings, neural networks have found their way into many commercial applications such as machine translation (Vaswani et al., 2017), virtual assistants (Van Den Oord et al., 2016), cancer detection (Ragab et al., 2019) and image generation Ramesh et al. (2022).

However, the utilisation of neural networks in many interesting settings is still infeasible. This is typically due to one of three main factors: a lack of suitable training data, the difficulty in training the model, or the cost of computational resources required. In this thesis we provide various techniques to help alleviate the latter two of the aforementioned issues. Specifically, introducing effective and easy to use training techniques for deep neural networks, including quantised neural

networks, which reduce the computational cost of deployment once trained. We draw inspiration from convex optimisation literature, and adapt these ideas for the deep learning setting.

1.2 Deep Neural Networks

Neural networks are parametric functions that map from some input space to a target space, where the type of each space is task dependent. In this thesis we will also use "weights" interchangeably to refer to the parameters of a neural networks. The freedom to choose the input and output spaces to represent different data types gives neural networks a lot of flexibility and allows them to encode functions useful for an extremely wide range of applications. For the task of image classification the input is the space of all images, possibly with a given height and width, and the output would be a vector representing a score for each of the image classes. Then an input image can be classified to the class with the largest respective output score. Alternatively, for monocular depth prediction the output space would instead be a single channel image of the same size as the input, where the intensity of each pixel encodes the distance from the camera at a given location. For audio classification the input would be a waveform, or possibly its Fourier transform, and the output would be a string of predicted characters or phonemes.

In order to allow neural networks to learn these highly complex and non-linear relationships, the function described by the neural network is designed to be extremely flexible. This is achieved by making neural networks "deep", specifically by stacking many layers that perform parameterised linear transformations and non-linear operations. Many versions of these simple building blocks have been proposed to help boost performance for different applications. In general increasing the number of parameters of a network, increases its expressive capacity and allows it to encode a more complex mapping from the input space to the output. This can be realised

by both using a larger number of layers, a larger number of weights in each layer, or most commonly, increasing both. Indeed, it has been proven that in the limit, when increasing the number of parameters, even shallow models with a few layers become universal function approximators (Hornik et al., 1989). Moreover, recent theory has shown that over-parameterisation is necessary if one wants to construct a function that interpolates the data smoothly (Bubeck and Sellke, 2021).

In practice, the trend over the last decade has been to stack increasing numbers of layers with evermore parameters to create deeper and deeper networks. These very large networks are capable of expressing almost any functional relationship over very large data sets. However, the increased size comes with additional computational and memory costs. Moreover, some of the largest models now have many billions of parameters and have a significant energy cost associated with them (Brown et al., 2020; Ramesh et al., 2022; Yuan et al., 2021). Hence, a key area of research is developing techniques to reduce the computational cost of deep networks. Many techniques have been proposed for achieving this (Neill, 2020), including using quantised parameters such as binary values (Gholami et al., 2021). In this thesis we consider deep neural networks, which for convenience we will often refer to as simply "neural networks".

1.3 Learning as Optimisation

Training a deep neural network is conventionally formulated as a mathematical optimisation problem (Goodfellow et al., 2016). The most popular paradigm is empirical risk minimisation, where one aims to minimise a scalar risk or loss function. The optimisation variable is a vector containing the parameters of the network to be trained. The loss function quantifies the risk or error of a prediction made over a set of training examples. Due to the highly non-convex nature of deep networks, and their large number of parameters in practice, finding an exact solution is not

feasible. Instead, a local minimum with low loss is deemed sufficient. However, this too is not the ultimate goal of supervised learning. Indeed, one really aims to find a model with good generalisation, that is, a model that makes accurate predictions on unseen examples. Optimising the performance on unseen examples is not directly possible. Instead a training set, which is assumed to be sampled from the same data generating distribution, is used.

Due to the over parameterised nature of deep neural networks it is often advantageous to complement the loss by a regularisation function. This function aims to penalise the complexity of the model and thus prevents overfitting and promote generalisation. We will refer to the sum of the loss and regularisation functions as the objective function or learning objective. In the majority of settings the parameters of the network are unconstrained or free to take any real value. However, this is not always the case and constraints can be applied to the weights of the network. This is typically done either as a different form of regularisation or when training a neural network with quantised parameters.

Many large networks can achieve, at least approximately, the minimum feasible loss value. For non-negative loss functions this is zero loss. If so, these models are said to interpolate the training set. We will refer to tasks where this property holds as the interpolation setting. As networks grow in size and capacity this setting becomes more likely on a given data set and hence it has become the focus of both empirical and theoretical research. The non-interpolation setting, by contrast, refers to cases where the network is unable to easily achieve zero training loss. This setting typically arises due to a relatively small network size compared to the complexity and size of the data set.

1.4 Optimisation Algorithms

Many iterative optimisation algorithms have been proposed to minimise the learning objective of deep neural networks. Due to the high dimensionality of the resulting optimisation problems, first order methods dominate. These methods use only first order information at each iteration, specifically, the loss and gradient of the loss with respect to the parameters. The computation and memory cost required in calculating and storing the Hessian and higher order derivatives makes exact second order methods infeasible. Stochastic optimisation techniques are best suited to the task due to the large data sets required for best results when training deep networks. These methods sample a small subset of the data set at each iteration to evaluate the gradient, hence giving a stochastic estimate at a lower computational cost.

The majority of neural network optimisation techniques were originally introduced in the optimisation literature for convex or non-convex optimisation problems. These algorithms typically come with provable convergence guarantees when applied to convex or smooth and non-convex functions. However, when training deep neural networks these guarantees rarely apply in practice. Instead, careful manual selection of hyperparameters is required to get good results. This is particularly true for stochastic gradient descent (SGD) (Robbins and Monro, 1951) and its derivatives, where the selection of the step size throughout training is critical. Even the same network architecture trained on different instances of the same task can require different schedules (He et al., 2016).

This deficiency and the dependence on good hyperparameters more generally, is one of the major obstacles in deployment of neural networks, as it increases the cost. This cost may take the form of manual tuning by a knowledgeable practitioner or extra computation by trying a large number of values. This is exacerbated further by the diversity of applications and specialist versions of neural networks that have been proposed. Hence, easy to use optimisation techniques that are widely applicable, are still an active area of research and form the focus of this thesis.

1.5 Thesis Outline and Contributions

Chapter 2 In Chapter 2 we review common techniques used for the training of deep feed-forward neural networks. We discuss popular algorithms such as stochastic gradient descent (Robbins and Monro, 1951), adaptive gradient methods and line search methods proposed for deep learning. We detail the situations in which these methods excel and highlight their shortcomings. We then talk about more recent optimisation algorithms developed for the interpolation setting including Stochastic Polyak Step (SPS) (Loizou et al., 2021), and ALI-G (Berrada et al., 2020). We focus particularly on ALI-G, as this method forms the inspiration for the work in Chapters 4 and 5.

Chapter 3 We introduce mathematical notation in Chapter 3. We formulate the standard supervised learning objective and related concepts. Additionally we introduce the mathematics behind many of the approaches previously discussed in Chapter 2.

Chapter 4 In Chapter 4 we propose a novel method for training deep neural networks that are capable of interpolation. At each iteration this method constructs a stochastic approximation of the learning objective. The approximation used is known as a bundle and is a pointwise maximum of linear functions. Our bundle contains a constant function that lower bounds the empirical loss. This enables us to compute an automatic adaptive learning rate, and thus an accurate solution. In addition, our bundle includes linear approximations computed at the current iterate and other linear estimates of the model parameters. The use of these additional approximations makes this method significantly more robust to its hyperparameters. Based on its desirable empirical properties, we term the method Bundle Optimisation for Robust and Accurate Training (BORAT). In order to operationalise BORAT, we design a novel algorithm for optimising the bundle approximation efficiently

at each iteration. We establish the theoretical convergence of BORAT in both convex and non-convex settings. Using standard publicly available data sets we provide a thorough comparison of BORAT to other single hyperparameter optimisation algorithms. Our experiments demonstrate that BORAT matches the state-of-the-art generalisation performance for these methods and is the most robust.

Chapter 5 We then turn our attention to the non-interpolating setting, that is, learning problems, where we do not expect to obtain zero or close to zero training loss. We introduce a novel algorithm, ALI-G+, which is based around the Polyak step size (Polyak, 1969). As we no longer have access to the optimal loss values *a priori*, this algorithm instead estimates these for each sample online. To realise this, we introduce a simple but highly effective heuristic for approximating the optimal value based on previous loss evaluations. Again we provide rigorous experimentation on a range of problems. From our empirical analysis we demonstrate the effectiveness of our approach, which outperforms other single hyperparameter optimisation methods on many standard benchmarks.

Chapter 6 In Chapter six we introduce quantised neural networks with a particular focus on binary neural networks (BNN) or networks that are designed so the majority of the parameters take the values -1 or 1. We additionally introduce what we will refer to as fully binary neural networks (FBNN) which have primarily binary activations as well, making them well suited to lightweight hardware. These networks cannot be trained using continuous optimisation algorithms due to their discrete weights and mostly zero gradients. Hence, the algorithms detailed in Chapter 2 are unsuitable for training BNN. In this chapter we detail existing methods for training quantised neural networks, such as the popular straight through estimator method (Courbariaux et al., 2015).

Chapter 7 In Chapter 7 we present a simple but effective method for training fully binary neural networks (FBNN) which we name Binary Networks the Easy Way, or BNEW for short. FBNN offer significant improvements in memory efficiency, energy usage, and inference speed over their floating point counterparts. Our approach to training FBNN splits the task into two phases. In the first phase, a model with binary activations and floating point weights is trained. In the second, a concave regulariser is added to encourage the weights to become binary. BNEW offers an alternative optimisation scheme to the straight through estimator that doesn't require an auxiliary set of weights during training. We show the effectiveness of BNEW by improving the state-of-the-art classification accuracy of a FBNN on the challenging ImageNet data set (Deng et al., 2009).

1.6 Publications

The work in this thesis has led to the following peer-reviewed publications:

- Work described in Chapter 4 resulted in the following publication:
  Alasdair Paren, Leonard Berrada, Rudra P. K. Poudel and M. Pawan Kumar. 2021. 'A Stochastic Bundle Method for Interpolating Networks'. *Journal of Machine Learning Research.*

- Work described in Chapter 5 resulted in the following publication:
  Alasdair Paren, Rudra P. K. Poudel and M. Pawan Kumar. 2022. 'Faking Interpolation Until You Make It'. *Transactions on Machine Learning Research.*

- Work described in Chapter 7 resulted in the following publication:
  Alasdair Paren and Rudra P. K. Poudel. 2022. 'Training Binary Neural Networks the Easy Way'. *British Machine Vision Conference 2022.*

Chapter 2

Related Work

2.1 Introduction

In this chapter we discuss existing techniques proposed for the training of deep neural networks. We start by introducing Stochastic Gradient Descent (SGD)(Robbins and Monro, 1951) as it forms the basis of many subsequent optimisation methods. SGD is still popular due to its strong generalisation performance. However, this performance is heavily dependent on the step size, also known as the learning rate, being adjusted during training. This is done through use of a manually selected step size or learning rate schedule. This deficiency makes SGD difficult to efficiently apply to new problems. We then discuss how subsequent optimisation techniques aimed to remove this requirement. These approaches can, for the most part, be grouped into three main categories; line search methods, adaptive gradient methods and Polyak-like step size methods.

2.2 Stochastic Gradient Descent

SGD was introduced by Robbins and Monro (1951) as an optimisation technique for monotone functions. SGD is a Stochastic version of the older gradient descent algorithm (GD), used for many convex and non-convex problems. GD is an itera-

tive method that only uses first order derivative information. Specifically, it uses the gradient of the learning objective calculated over the entire training set. At each step GD simply selects the next iterate to be along the ray described by the negative gradient direction, where the gradient is evaluated at the current iterate. The distance in this direction is given by the present step size multiplied by the magnitude of the gradient. The computation of the objective's gradient scales linearly with the size of the training set. So while GD works well for problems with a small training set, the cost of each step becomes increasingly expensive as the data set inceases in size. Empirical results are nearly always improved by using a larger data set that samples more of the relevant input space. To this end data augmentation schemes that artificially increase the size of the training set by orders of magnitude are commonplace. This makes the per iteration cost of GD infeasible in most real world settings.

Stochastic Gradient Descent (SGD), in contrast, does not evaluate the gradient of the objective calculated over the entire training set at each iteration. A small subset or mini-batch of the training set is sampled and used instead. This gives a stochastic estimate of the gradient. Importantly, the iteration time no longer depends on the size of the training set so any training set size can be used, without affecting the run time. If a large training set is used, each example is just sampled less frequently. The iteration cost of SGD scales linearly with the mini-batch size and trades off the stochasticity of this estimate against the computational and memory cost of its calculation. In practice, the batch size is typically determined by the hardware used for training. It is common to use the largest size that fits into the hardware memory. However, using a smaller batch size can often produce superior results, as the noise in the gradients can help the algorithm escape bad local minima in the learning objective. All deep neural network optimisation techniques that have seen wide success follow this approach and use a mini-batch at each iteration to calculate stochastic estimates of information of the objective. When minimising

an optimisation problem constrained to a convex set SGD is modified to include a projection back onto the feasible set after each update. This algorithm is known as Projected Stochastic Gradient Descent (PSGD).

SGD and PSGD have been used to produce state-of-the-art generalisation performance for many supervised learning tasks. For established learning problems using deep feed forward networks, once effective step size schedules have been found, SGD is rarely outperformed by other optimisation techniques. However, the downside of SGD is that it requires the manual design and refinement of a learning rate schedule for best performance. Many forms of schedule have been proposed in the literature, including piecewise constant (Huang et al., 2017), geometrically decreasing (Szegedy et al., 2015), and warm starts with cosine annealing (Loshchilov and Hutter, 2017). Consequently, practitioners who wish to use SGD in a novel setting need to find a good schedule to use for their learning task. To that end, they first need to choose the parameterisation of the schedule and then tune the corresponding hyperparameters. For example, a piecewise linear scheme requires an initial learning rate value, a decay factor, and a list of times or metric to determine at which points in training to decay the learning rate. This results in a large search space which increases exponentially in combination with other problem dependent quantities, such as constants controlling regularisation or batch size. As SGD can be sensitive to these hyperparameters, and their optimal values often are highly interdependent, the resulting cross-validation scheme necessary for best results can be prohibitively expensive.

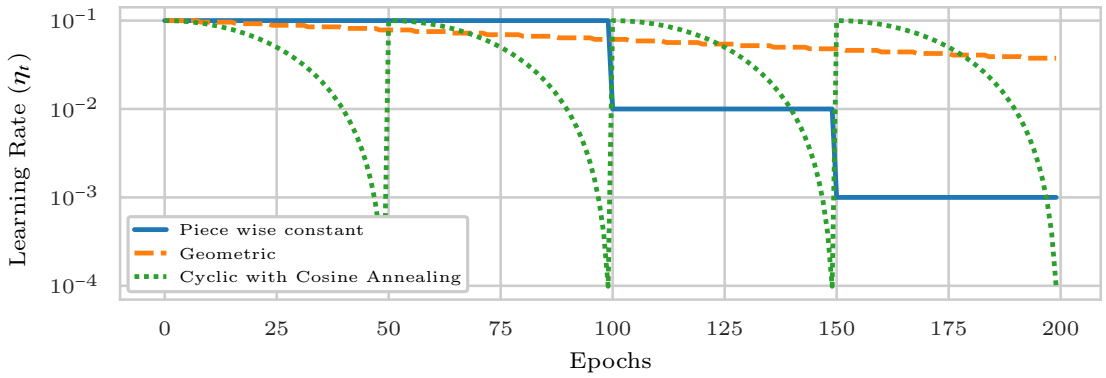


Figure 2.1: *SGD learning rate schedules proposed in the literature*

2.3 Momentum

SGD and related methods are commonly used in combination with momentum, either "Nesterov momentum" or "Polyak momentum". When using Polyak momentum the last update vector is stored. At each step a convex combination of the stored and the newly calculated updates is applied. The weighting of these two vectors is a hyperparameter that must be selected before training. A weighting of 0 would correspond to no momentum, while a weighting of 0.9 for the previous step is commonly used. This technique is named momentum due to the stored update sharing similarity to the momentum of a ball rolling around the loss landscape. Alternatively, momentum can be thought of as a computationally cheap method for reducing the noise from mini-batching in each update. The gradient calculated on a mini-batch can have high variance particularly when the batch size is small. Momentum entails using an exponential moving average of recent gradients for the update rather than a single evaluation, which helps reduce this noise. Nesterov momentum (Nesterov, 1983) modifies this idea so the gradient is evaluated at the iterate with the momentum already applied. Nesterov momentum boasts an optimal convergence rate for convex problems. However, for deep learning problems where no such proof exists both variants are still popular.

2.4 Line Search Methods.

Line search methods use perhaps the most obvious alternative to avoid manually selecting a step size schedule. As their name suggests, these approaches run a line search algorithm at each step to automatically pick a step size. A linear search algorithm performs a search procedure along the ray in the negative gradient direction rather than using the pre-selected step size of SGD. A series of trial points are sampled using a heuristic. A trial point is selected to be the next iterate if it satisfies a criterion. A number of different variants have been suggested in the literature, depending on the exact heuristic and criterion used.

Vaswani et al. (2019) present algorithms based on the Armijo and Goldstein line search methods (Armijo, 1966), classically used for deterministic gradient descent. They also introduce heuristics with the aim of minimising the number of extra forward passes required, which they claim reduces the average number required to one extra per batch. While these methods provide strong guarantees for convex problems they can be temperamental training deep neural networks.

Mutschler and Zell (2020) and Hao et al. (2021) present a modification to standard line search methods. Instead of performing a linear search, these works assume the loss function is approximately parabolic in the negative gradient direction and aim to fit a quadratic approximation. Mutschler and Zell (2020) provide empirical justification for this approximation. With this assumption only a single extra loss and gradient is needed to fit a quadratic model. Hence only one extra loss and gradient evaluation is required. Hao et al. (2021) detail a variant that samples the loss at many points along the negative gradient direction, and then fits a quadratic model using least squares. In both cases once the approximation has been constructed and has positive curvature it can be minimised in closed form. If however the curvature of the loss in negative gradient direction is negative, a fixed step size is used, or the mini-batch is re-sampled.

Line search methods can present strong performance, however, they have two

major deficiencies. First, line search methods require additional computation per batch over typical first order methods, resulting in a longer training time. Second, while removing the need for a step size schedule they invariably introduce extra hyperparameters governing the heuristics on how trial points are selected or whether a trial point is accepted as the next iterate. While these hyperparameters are fixed over training and do typically not require a schedule they must be tuned per experiment for best results. Poorly selected hyperparameters can result in many extra loss evaluations being required thereby increasing the run time significantly or resulting in poor optimisation performance.

2.5 Adaptive Gradient Methods

Adaptive Gradient methods also aim to remove the need for a learning rate schedule. However instead of searching ahead in parameter space these approaches use heuristics based on previous gradient evaluations to scale a single fixed learning rate for each parameter independently. Many Adaptive Gradient methods have been proposed with slightly different heuristics, such as Adagrad (Duchi et al., 2011), Adam (Kingma and Welling, 2014) or more recently Adabound (Luo et al., 2019), RMSPROP (Tieleman and Hinton, 2012) and many other variants (Zeiler, Zeiler; Orabona and Pál, 2015; Défossez and Bach, 2017; Levy, 2017; Mukkamala and Hein, 2017; Zheng and Kwok, 2017; Bernstein et al., 2018; Chen and Gu, 2018; Shazeer and Stern, 2018; Zaheer et al., 2018; Chen et al., 2019; Loshchilov and Hutter, 2019; Luo et al., 2019; Heo et al., 2021). These algorithms are easy to use as they require a single fixed learning rate hyperparameter to be selected that tends to provide decent results over a wide range of values (Sivaprasad et al., 2020). However, once tuned, other optimisation algorithms such as SGD or line search methods provide superior generalisation performance over Adaptive Gradient methods on a wide range of supervised learning benchmarks (Berrada et al., 2020; Wilson et al., 2017).

2.6 Methods for Interpolation

As neural networks have increased in size, the interpolation property, as described in Chapter 1, has become more common on small a medium sized data sets. This property was initially utilised by early efforts to analyse the convergence speed of SGD. In the interpolation setting Ma et al. (2018); Vaswani et al. (2019) and ,Zhou et al. (2019) demonstrate that SGD achieves the convergence rates of full-batch gradient descent.

In the interpolation setting, it is known *a priori* that the optimal function value is simply the minimum possible function value. Hence, for all non-negative loss functions zero loss will be reached. Moreover, it implies that the minimum function value must be achieved for all samples simultaneously. This allows a Polyak-like step size to be used (Polyak, 1969). The Polyak step size selects the next iterate to be at the point where the ray in the negative gradient direction intercepts the hyperplane corresponding to zero loss. Without any modification this approach does not perform well for highly non-convex deep learning tasks, as it can result in very large updates to the parameters. However, this issue can be remedied by use of a fixed maximal step size, which reduces the magnitude of all updates to below this value.

Berrada et al. (Berrada et al., 2020) introduced the ALI-G algorithm specifically designed for settings where interpolation holds. At each step ALI-G solves an interpolation aware proximal problem. The closed form solution is a Polyak-like step size with a maximal learning rate, which gives a strong justification for a step size of this form. When applied to common deep learning benchmarks the ALI-G algorithm produces strong empirical results. Specifically, ALI-G outperforms all Adaptive Gradient Methods, almost achieving the performance of SGD with a bespoke learning rate schedule, without requiring one itself. Alongside this empirical success, Berrada et al. provide convergence bounds of ALI-G applied to convex problems and a special set of non-convex problems. We give extra detail on the

ALI-G algorithm in Chapter 3 as it forms the foundation for the ideas explained in Chapters 4 and 5. Loizou et al. (2021) present a very similar algorithm to ALI-G which they name Stochastic Polyak Step (SPS). The SPS step size does not have a clear derivation and has a slightly different form to that of ALI-G, containing two hyperparameters rather than one. However, this difference allows theoretical results with milder assumptions to be established for SPS. Out of SPS and ALI-G once properly tuned ALI-G offers slightly better empirical performance on standard benchmarks (Berrada et al., 2021). In their work Stochastic Gradient Descent with Polyak's Learning Rate Oberman and Prazeres (2019) proposed a Polyak-like Step Size for which they provide some theoretical results. However, in order to estimate the minimum loss they suggest training the model with tuned SGD first, which limits is practicality. The L_4 algorithm (Rolinek and Martius, 2018) instead uses a modified version of the Polyak step size. However, the L_4 algorithm computes an online estimate of the minimum loss value rather than relying on the interpolation property. This requires three hyperparameters, which are sensitive to noise and crucial for empirical convergence of the method. In addition, L_4 does not come with convergence guarantees. S Finally, Liu et al. (2019) propose to exploit interpolation to prove convergence of a new acceleration method for deep learning. However, their experiments suggest that the method still requires the use of a hand-designed learning rate schedule.

Many of the optimisation methods for interpolation, such as ALI-G and SPS remove the need for a learning rate schedule while retaining performance similar to SGD. This makes these algorithms far more easily applied in new settings where a good learning rate schedule is not known in advance. ALI-G is perhaps the best for new settings, as it only requires a single maximal learning rate parameter to be selected. However these methods can still be costly to tune when there are other task specific parameters. Finally, these methods rely on the interpolation property thus it does not make sense to use them outside this setting.

Chapter 3

Preliminaries

3.1 Learning Task

As described in Chapter 1, a neural network is a parameterised mapping $g : \mathcal{H} \to \mathcal{Y}$ where $\mathcal{H} \in \mathbb{R}^n$ denotes a task specific "input" space and $\mathcal{Y} \in \mathbb{R}^m$ denotes a relevant "output" space. In this thesis we will use $\boldsymbol{w} \in \mathbb{R}^d$ to represent the vector containing the parameters of the network. The goal of training a neural network is to find a setting of $\boldsymbol{w}$ such that for certain inputs $\boldsymbol{x}_i$ the network produces a desired output $\boldsymbol{y}_i$, where $\boldsymbol{y}_i = g(\boldsymbol{x}_i, \boldsymbol{w})$.

For most machine learning tasks we are only interested in some subset of the input space $\boldsymbol{x} \in \mathcal{X} \subset \mathcal{H}$. For example, while the input space $\mathcal{H}$ could be the space of all images for autonomous driving applications, one might only be interested in street scenes. Mathematically defining any subset $\mathcal{X}$ in a rigorous way is highly subjective and almost impossible. Instead, a finite set of test examples $\{(\boldsymbol{x}_j^{test}, \boldsymbol{y}_j^{test}) : j \in 1, ..., J\}$ is used to measure performance.

Hence, the goal of the learning task is to find a $\boldsymbol{w}$ that achieves strong performance on this test set. To quantify performance, a task specific scalar function $h(\boldsymbol{y}_i, \boldsymbol{y}_i^{test})$ is used. For classification tasks h is commonly accuracy which gives a score of 1 if the class denoted by $\boldsymbol{y}_i$ matches that of $\boldsymbol{y}_i^{test}$ and 0 otherwise. With-

out loss of generality we will assume larger values of h imply better performance on the chosen tasks. In other words, we want to maximise $\mathbb{E}_{j\in\mathcal{J}}[h(\boldsymbol{y}_j, \boldsymbol{y}_j^{test})]$ where $\mathcal{J}$ is a distribution, typically uniform, over $\{1, ..., J\}$. It is not possible to optimise a loss calculated on unseen data, thus a training set is used $\{\boldsymbol{x}_z^{train}, \boldsymbol{y}_z^{train}\}$ for $z \in \{1, ..., Z\}$, which is assumed to have been produced by the same data generating distribution as the test set. It is common to make use of an additional validation set to provide an estimate of the test performance during training.

3.2 Loss Function

Many performance metrics h are unsuitable for direct optimisation, even when calculated over the training set, as they have zero gradient almost everywhere and are discontinuous. Hence, a loss function with more desirable properties is used. A common example of this is classification accuracy h, which cannot easily be directly optimised. Instead, a loss function such as the Cross Entropy (CE) or multi-class hinge loss is used. We define $\ell_z(g(\boldsymbol{x}_z, \boldsymbol{w}), \boldsymbol{y}_z)$ to be the loss function for the z^{th} training example. For ease of notation we will drop the dependence of ℓ_z on the $\boldsymbol{x}_z$ and $\boldsymbol{y}_z$ and just write ℓ_z as a function of the parameters of the network $\ell_z(\boldsymbol{w})$.

We assume that each ℓ_z admits a known lower bound B. For the vast majority of loss functions used in machine learning, such as cross-entropy, hinge losses, or norm based losses for regression, the lower bound is $B = 0$. For a loss function where B is not zero, simply adding a constant $-B$ to ℓ_z, results in a lower bound of 0. We can now define the loss over the training set as an expectation over $z \in \mathcal{Z}$:

$$f(\boldsymbol{w}) \triangleq \mathbb{E}_{z\in\mathcal{Z}}[\ell_z(\boldsymbol{w})]. \tag{3.1}$$

3.3 Regularisation

It is often desirable to encourage generalisation by use of a non-negative regularisation function $R_\lambda(\boldsymbol{w})$. The typical choices for R are $\frac{\lambda}{2}\|\boldsymbol{w}\|^2$, $\frac{\lambda}{2}\|\boldsymbol{w}\|_1$, or some combination of the two. In both cases λ is a hyperparameter that governs the strength of the regularisation, where $\lambda = 0$ corresponds to no regularisation being used. In this thesis $R_\lambda \triangleq \frac{\lambda}{2}\|\boldsymbol{w}\|^2$ unless stated otherwise.

3.4 Problem Formulation

The task of training a neural network can be expressed as the optimisation problem $(\mathcal{P})$. Or in other words we aim to find a feasible vector of parameters $\boldsymbol{w}_\star \in \Omega$ that minimizes f:

$$\boldsymbol{w}_\star \in \operatorname*{argmin}_{\boldsymbol{w}\in\Omega} f(\boldsymbol{w}) + R_\lambda(\boldsymbol{w}), \tag{$\mathcal{P}$}$$

Where Ω the feasible region expressing any constraints on the parameters of the network. For unconstrained problems, one can set $\Omega = \mathbb{R}^d$. We refer to the minimum value of f over Ω as $f_\star$: $f_\star \triangleq \min_{\boldsymbol{w}\in\Omega} f(\boldsymbol{w})$.

3.5 Interpolation

In Chapter 4 we consider problems that satisfy the interpolation property, as described in Chapter 1, which we formally introduce here. For neural networks that can interpolate the data, we assume the following property holds:

$$\exists\, \boldsymbol{w}_\star :\ \forall z \in \mathcal{Z}, \ell_z(\boldsymbol{w}_\star) \leq \epsilon, \tag{3.2}$$

where ϵ is a tolerance on the amount of error in the interpolation assumption. We will often want to make reference to the case when $\epsilon = 0$. Following previous work (Ma et al., 2018) we will refer to this setting as perfect interpolation. The interpo-

lation property is satisfied in many practical cases, since modern neural networks are typically trained in an overparameterised regime, where the parameters of the model far exceed the size of the training data (Li et al., 2020). Additionally, most modern DNN architectures can be easily increased in size and depth, allowing them to interpolate all but the largest data sets. Note, the data have to be labelled consistently for this to be possible. For instance, it is impossible to interpolate a data set with two instances of the same image that have two different labels.

3.6 First Order Methods

Stochastic first order iterative algorithms are currently the best approach for finding a good solution to problem $(\mathcal{P})$, and hence training deep neural networks. Moreover, all optimisation algorithms discussed in this thesis will be of this type. In this section we briefly introduce these methods and some useful notation. These approaches start at a starting point $\boldsymbol{w}_0$, which corresponds to the parameters of the neural network after some initialisation scheme. For example, sampling from a zero mean Gaussian where standard deviation depends on the location of the parameter in the network (He et al., 2015). Once initialised, an update to the parameters for the network is repeatedly applied for T steps or until some stopping condition is reached. We will use the subscript t to denote the time step or iteration number. At each step a mini-batch of the training set is selected by sampling b indexes $\boldsymbol{z} \in \mathcal{Z}$ where b is the batch size. Thus we use z_t to denote the mini batch sampled at time t. This mini batch is then used to calculate $\ell_{z_t}(\boldsymbol{w})$ and its gradient with respect to $\boldsymbol{w}_t$, which we denote with $\nabla \ell_{z_t}(\boldsymbol{w}_t)$. In practice, calculating these quantities involves performing a "forward pass" to compute $\ell_{z_t}(g(\boldsymbol{x}_{z_t}, \boldsymbol{w}), \boldsymbol{y}_{z_t})$. This is followed by a "backwards pass", where backpropagation is used to compute $\nabla \ell_{z_t}(\boldsymbol{w}_t)$.

3.7 Proximal Perspective of SGD and PSGD

SGD and the related PSGD algorithm have insightful interpretations which help explain the importance of its step size, and set the stage for the more complex ALI-G optimiser (Berrada et al., 2020). The PSGD algorithm can be seen as solving a sequence of proximal problems. Within each proximal problem, a minimisation is performed over an approximate local model of the loss. This approximation is the first order Taylor's expansion of ℓ_{z_t} around the current iterate and a proximal term. At time step t, this proximal problem has the form:

$$\boldsymbol{w}_{t+1} = \underset{\boldsymbol{w}\in\Omega}{\operatorname{argmin}} \left\{ \frac{1}{2\eta_t}\|\boldsymbol{w} - \boldsymbol{w}_t\|^2 + \ell_{z_t}(\boldsymbol{w}_t) + \nabla\ell_{z_t}(\boldsymbol{w}_t)^\top(\boldsymbol{w} - \boldsymbol{w}_t) \right\}. \tag{3.3}$$

Here η_t is the learning rate at time t. For convex Ω, problem (3.3) can be solved in two steps: first solving the unconstrained problem and then projecting $\boldsymbol{w}$ onto Ω using Euclidean Projection. Setting $\Omega = \mathbb{R}^d$ removes the need for projection and we recover SGD. To solve the unconstrained problem one can find the point at which the gradient is zero, via differentiation, to recover the following closed form update:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t\nabla\ell_{z_t}(\boldsymbol{w}_t). \tag{3.4}$$

In practice, this update is often used in conjunction with momentum as described in Chapter 2. Nesterov momentum (Nesterov, 1983) gives rise to the following updates:

$$\boldsymbol{m}_0 = 0 \tag{3.5}$$

$$\boldsymbol{m}_t = \mu\boldsymbol{m}_{t-1} - \eta_t\nabla\ell_{z_t}(\boldsymbol{w}_t), \tag{3.6}$$

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \eta_t\nabla\ell_{z_t}(\boldsymbol{w}_t) + \mu\boldsymbol{m}_t. \tag{3.7}$$

Here μ is the momentum hyperparameter, and is typically set to $\mu = 0.9$. When Polyak momentum is used instead equation (3.7) is replaced with $\boldsymbol{w}_t = \boldsymbol{w}_t - \eta_t\boldsymbol{m}_t$.

3.8 Adaptive Moment Estimation (Adam)

The Adam optimiser (Kingma and Ba, 2015) is probably the most popular Adaptive Gradient Method (Chen et al., 2019). Adam has found wide success due to its lack of sensitivity with respect to its hyperparameters, and the broad range of settings where it produces good results. Adam has found particular success for settings where there is no clear objective being minimised, such as training GANs and Q-learning with function approximation (Wilson et al., 2017).

The Adam optimiser is an adaptive gradient method; that is, it uses previous gradient information to inform the step size with the aim to remove the need for a learning rate schedule. However, Adam often produces best results when used with a decreasing learning rate so use of a schedule is still common in practice.

Adam scales a global step size independently for each parameter. This gives it an advantage over methods such as SGD that uses the same step size for all parameters. This is especially true when the magnitude and variance of the gradients for each parameter are significantly different, or where gradients are sparse. Adam effectively normalises the magnitude of updates, giving larger step size to parameters which have historically had smaller gradients. This is achieved by keeping track of a running average of the first and second moments of the gradients for each parameter. The first moment is used functionally similar to Polyak Momentum (Section 3.7). The inverse of the square root or second moment is used to scale the learning rate.

Mathematically, the Adam update can be expressed as follows:

$$\hat{\boldsymbol{m}}_t = \frac{\boldsymbol{m}_t}{1-\beta_1^t}, \tag{3.8}$$

$$\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\nabla \ell_{z_t}(\boldsymbol{w}_t), \tag{3.9}$$

$$\hat{\boldsymbol{v}}_t = \frac{\boldsymbol{v}_t}{1-\beta_2^t}, \tag{3.10}$$

$$\boldsymbol{v}_t = \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\nabla \ell_{z_t}(\boldsymbol{w}_t)^2, \tag{3.11}$$

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \frac{\eta}{\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon}\hat{\boldsymbol{m}}_t, \tag{3.12}$$

where η is the learning rate hyperparameter and $\beta_1, \beta_2, \epsilon$ are constants, which are typically set to the following values $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$.

Since Adam's inception a number of variants have been proposed. To name a couple AdamP (Heo et al., 2021) suggest a modification of Adam designed for optimising networks including batch-norm layers (Ioffe and Szegedy, 2015). Whereas Loshchilov and Hutter (2019) claim to provide a better method of applying regularisation when using Adam. A number of works have scrutinised Adam's theoretical performance. Wilson et al. (2017) and Reddi et al. (2018) both present compelling arguments against the use of Adam. These works provide simple convex optimisation problems where Adam does not converge to the optimal solution. Since these results, significant work has been put into proving theoretical guarantees for Adam. Chen et al. (2019) and Défossez et al. (2020) detail convergence proofs for Adam in stochastic non-convex and convex settings respectively. However these results require η to vary as a function of β_1 and β_2 which is not suggested in the original paper. Adam's empirical performance has also been called into question. Reddi et al. (2018) show that on many standard supervised learning benchmarks, non-adaptive methods such as SGD produce superior generalisation performance over Adam and other adaptive gradient methods. However Sivaprasad et al. (2020), instead suggest when tuning is limited, Adam provides superior results over SGD. We note that these two points of view do not really contradict each other as the work

of Reddi et al. (2018) assumed that all methods had been tuned and had access to good hyperparameters. Despite these criticisms, the original Adam algorithm remains popular due to its ease of use.

3.9 Adaptive Learning-rates for Interpolation with Gradients (ALI-G)

Berrada et al. (2020) consider problems of type $(\mathcal{P})$ where the interpolation property (3.2) holds and they propose a Polyak-like step size for this setting. Figure 3.1 gives a geometric representation of the Polyak step size when applied to a function $f(\boldsymbol{w})$ with $f_\star = 0$. The ALI-G step size is derived by identifying a deficiency for the

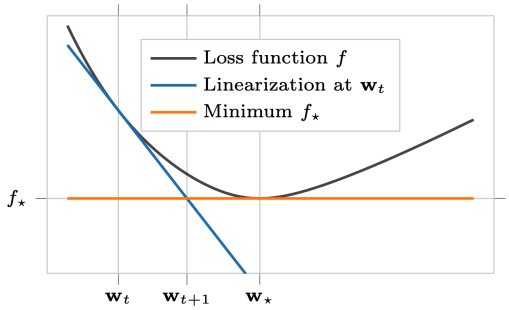


Figure 3.1: *Illustration of the Polyak step size in 1D. In this case, and further assuming that $f_\star = 0$, the algorithm coincides with the Newton-Raphson method for finding roots of a function (Berrada et al., 2020)*

interpolating setting in the SGD proximal problem, as detailed in Equation (3.3). We know in this setting ℓ_z will approach zero. As this happens the approximation of the loss inside each proximal minimisation permits negative values even though the loss for $(\mathcal{P})$ is defined to be non-negative. Specifically when $\frac{\eta}{2}\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 \geq \ell_{z_t}(\boldsymbol{w}_t)$ the value of the minimum of (3.3) will be negative. Thus, Berrada et al. (2020) propose to address this issue by altering (3.3) by taking the pointwise maximum of the lower bound 0 and the linear approximation of ℓ_z. Hence, the ALI-G proximal

problem can be expressed as:

$$\boldsymbol{w}_{t+1} = \underset{\boldsymbol{w}\in\Omega}{\operatorname{argmin}} \left\{ \frac{1}{2\eta}\|\boldsymbol{w} - \boldsymbol{w}_t\|^2 + \max\{0, \ell_{z_t}(\boldsymbol{w}_t) + \nabla\ell_{z_t}(\boldsymbol{w}_t)^\top(\boldsymbol{w} - \boldsymbol{w}_t)\} \right\}. \quad (3.13)$$

This minimisation is non-smooth and thus is best solved in the dual. Constructing the Lagrange dual gives:

$$\underset{\alpha\in[0,1]}{\operatorname{argmax}} \left\{ -\frac{\eta}{2}\|\alpha\nabla l_{z_t}(\boldsymbol{w}_t)\|^2 + \alpha l_{z_t}(\boldsymbol{w}_t) \right\}, \quad (3.14)$$

where α is the dual variable. This dual problem is a maximisation over a constrained concave function in one dimension. Hence, one can obtain the optimal point by projecting the unconstrained solution onto the feasible region. This results in the following closed form solution:

$$\gamma_t = \min \left\{ \frac{l_{z_t}(\boldsymbol{w}_t)}{\eta\|\nabla l_{z_t}(\boldsymbol{w}_t)\|^2}, 1 \right\}. \quad (3.15)$$

From the Karush–Kuhn–Tucker (KKT) conditions, Berrada et al. (2020) recover the following update:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \gamma_t\eta\nabla l_{z_t}(\boldsymbol{w}_t). \quad (3.16)$$

Specifically, the ALI-G step size automatically scales down a maximal learning rate η to an appropriate value by a factor $\gamma_t \in [0, 1]$. The ALI-G update can be viewed as a stochastic analogue of the Polyak step size (Polyak, 1969) with the addition of a maximal value η. Berrada et al. (2020) often achieve best results when ALI-G is used in conjunction with a Nestorov momentum as described in equations (3.6) and (3.7).

3.9.1 Regularisation

In order to ensure that interpolation holds, Berrada et al. (2020) incorporate regularisation as a constraint on the feasible domain rather than via use of a regularisation function $R_\lambda(\boldsymbol{w})$. Specifically, they define: $\Omega = \left\{\boldsymbol{w} \in \mathbb{R}^d : \ \phi(\boldsymbol{w}) \leq r\right\}$ for some value of r. In the deep learning setting, this allows the objective function to be driven close to zero without unrealistic assumptions about the value of the regularisation term for the final set of parameters. This framework can handle any constraint set Ω on which Euclidean projections are computationally efficient. This includes the feasible set induced by ℓ_2 regularisation: $\Omega = \left\{\boldsymbol{w} \in \mathbb{R}^d : \ \|\boldsymbol{w}\|_2^2 \leq r\right\}$, for which the projection is given by a simple rescaling of $\boldsymbol{w}$. If it is desired not to use regularisation, one can define $\Omega = \mathbb{R}^d$, and the corresponding projection is the identity.

3.9.2 Performance

The ALI-G update is computationally cheap with the evaluation of $\|\nabla l_{z_t}(\boldsymbol{w}_t)\|^2$, and the projection onto Ω being the only extra computation required over SGD. ALI-G does not require a learning rate schedule and produces strong generalisation performance on a number of supervised learning tasks (Berrada et al., 2020). Thus, it offers a competitive alternative to SGD. However, it has two major shortcomings. First, it is sensitive to its hyperparameters and typically requires the learning rate η, batch size b and feasible radius r to be tuned for best results. Second, it can only be used when the interpolation property holds. Addressing these two deficiencies forms the inspiration for the work in Chapters 4 and 5 respectively.

Chapter 4

Bundle Optimisation for Robust and Accurate Training (BORAT)

4.1 Introduction

As discussed in Section 3.9, the ALI-G optimiser (Berrada et al., 2020) has many desirable properties. In this chapter, inspired by ALI-G, we introduce a novel method for training deep neural networks that are capable of interpolation. ALI-G can be sensitive to hyperparameters such as its maximial step size, the mini-batch size, and task regularisation amount. Our method helps alleviate this weakness of ALI-G and offers reduced sensitivity to its hyperparameters. This increased robustness is achieved by using a more complex model of the loss within each proximal problem. The choice of this model results in an optimiser that requires far less tuning of its hyperparameters while achieving comparable generalisation performance to algorithms such as ALI-G or SGD with a refined learning rate schedule. This is particularly evident when applied to challenging non-smooth losses. Consequently this leads to a reduction in the time, cost, and energy required when finding a high accuracy network for a new task. Additionally, our method also produces superior results over ALI-G in the presence of label noise.

A key insight of our approach is the observation that ALI-G's approximation of the loss is a bundle of size two; or specifically the pointwise maximum of two linear functions. This leads to the natural extension of using bundle sizes larger than two. We consider the general case where the pointwise maximum over a small number of stochastic linear approximations is used to model the loss, where N is used to denote the number of linear approximations used. Similar to ALI-G, when using a bundle of size N, we utilise one linear approximation to enforce the lower bound on the loss $0 = \min_{\boldsymbol{w}} \ell(\boldsymbol{w})$. This enables us to automatically adapt the magnitude of each update to an appropriate value. The remaining $N - 1$ approximations, are used to better approximate the loss, making our method significantly more robust to its hyperparameters. Based on its desirable empirical properties, we term our method Bundle Optimisation for Robust and Accurate Training (BORAT). In order to operationalise BORAT, we design a novel algorithm for optimising the bundle approximation efficiently at each iteration. We establish the theoretical convergence of BORAT in both convex and non-convex settings. Using standard publicly available data sets, we provide a thorough comparison of BORAT to other single hyperparameter optimisation algorithms. Our experiments demonstrate that BORAT matches the state-of-the-art generalisation performance for these methods and is more robust.

4.2 Bundle Methods

Bundles and Bundle Methods have been previously proposed for the optimisation of non-smooth convex functions (Lemaréchal et al., 1995; Smola et al., 2007; Auslender, 2009). However, these works do not treat the stochastic case, as they consider small problems where the full gradient can be cheaply evaluated. Asi and Duchi (2019) introduced a bundle method as part of their family of proximal stochastic optimisation algorithms for convex problems. However, their work did not focus

on the training of neural networks. Hence, to our knowledge, BORAT is the first proposed bundle method specifically for the optimisation of neural networks. Additionally, we note the following weaknesses of the bundle version of the Aprox family, introduced by Asi and Duchi (2019). First, their method is assumed to have an exponentially decaying learning rate schedule, and even in the interpolating convex setting requires two hyperparameters. Second, Asi and Duchi (2019) do not provide details on how to solve each proximal problem efficiently.

4.3 The BORAT Algorithm

In this section we detail the BORAT algorithm. We start by introducing BORAT's proximal problem that is exactly solved at each iteration. We explain its advantages and disadvantages in relation to the SGD and ALI-G proximal problems. Each BORAT proximal problem is best solved in the dual. Hence, we next introduce the dual problem, which permits a far more efficient solution due to its low dimensionality. When more than two linear approximations are used within a bundle a closed form solution is no longer possible. Hence, we next detail our novel direct method for efficiently solving the dual problem arising. This algorithm exploits the small size of the bundle to compute the exact optimum by solving a finite number of linear systems, removing the need for an inner iterative optimisation algorithm.

4.3.1 Advantages of Bundles with More Than Two Pieces

While ALI-G has many favourable qualities, its local model of the loss is still crude. Using extra approximations allows us to model more variation over the parameter space and positive curvature of the loss. The main disadvantage of a larger bundle is it requires multiple gradient evaluations to be performed and then held in memory, and does not permit a closed form solution. We next give two motivating examples to demonstrate why using a more complex model of the loss is worth these drawbacks

and in general leads to increased robustness with respect to the algorithm's maximal step size η.

Any convex function can be perfectly modelled by the pointwise maximum over an infinite number of linear approximations. While intractable, performing a minimisation over this model could recover the true optimum by definition. With this perfect model any value of η could be used. Indeed, setting η to a large enough value would recover the optimum in a single step. This example demonstrates that, at least asymptotically, as the accuracy of the local model increases we can expect a reduced dependence on the correct scale of the step size.

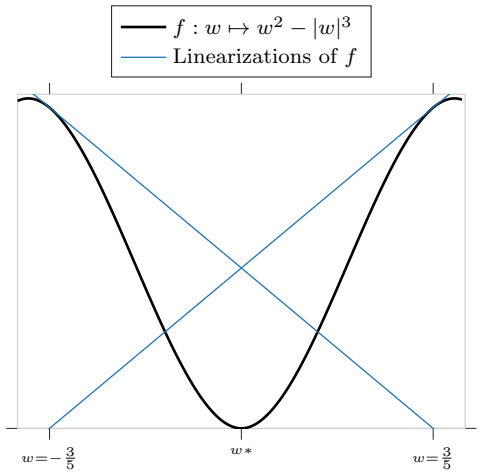


Figure 4.1: *A simple example where the ALI-G step size oscillates due to non-convexity. For this problem, ALI-G only converges if its maximal learning rate η is less than* 10*. By contrast, for the same example BORAT with $N > 2$ converges for all values of η. Additionally, for $\eta \geq 10$ it converges to the optimum in a single update.*

Figure 4.1 provides a non-convex motivating example for use of larger values of N. Here we demonstrate a 1D symmetric function where ALI-G does not converge for large η. Instead it oscillates between the two values $w = -3/5$ and $w = 3/5$. However, if we were to use a bundle with a $N \geq 3$ our model of the loss would include both the linear approximations at $w = -3/5$ and $w = 3/5$ simultaneously and hence when minimising over this model we converge to the optimum for any value of η. While this is a carefully constructed synthetic example, it highlights why

we expect a more accurate model of the loss to help reduce the dependence on the step size.

4.3.2 Primal Problem

Similar to SGD and ALI-G, BORAT exactly solves a proximal optimisation problem calculated over an approximation of the loss at every iteration. BORAT's approximation is a bundle of size N, or in other words the pointwise maximum over N linear functions. Each linear approximation of the loss is constructed at a point $\hat{\boldsymbol{w}}_t^n$ using a different stochastic evaluation of the loss function, denoted as $\ell_{z_t^n}$. The subscript t indicates the iteration number and the superscript n indexes over the N linear approximations. As before, the subscript z indexes the training example or examples in the batch at time t. Thus, the BORAT proximal problem N can be expressed as:

$$\boldsymbol{w}_{t+1} = \underset{\boldsymbol{w}\in\Omega}{\operatorname{argmin}} \left\{ \frac{1}{2\eta}\|\boldsymbol{w} - \boldsymbol{w}_t\|^2 + \max_{n\in[N]}\{\ell_{z_t^n}(\hat{\boldsymbol{w}}_t^n) + \nabla\ell_{z_t^n}(\hat{\boldsymbol{w}}_t^n)^\top(\boldsymbol{w} - \hat{\boldsymbol{w}}_t^n)\} \right\}, \quad (4.1)$$

where we use the notation $[N]$ to define the set of integers $\{1, ..., N\}$ and $\Omega = \left\{\boldsymbol{w} \in \mathbb{R}^d : \ \|\boldsymbol{w}\|_2^2 \leq r\right\}$. We now show how to convert (4.1) into the more convenient form of $\max_{n\in[N]} \left\{\boldsymbol{a}^{n\top}(\boldsymbol{w} - \boldsymbol{w}_t) + b^n\right\}$. Within a bundle, each linear approximation is formed around a different point $\hat{\boldsymbol{w}}_t^n$. Hence, in order to write each linear approximation in the aforementioned compact form we split each linear term in two. The first piece is a constant term, that does not depend on $\boldsymbol{w}$, and is a multiple of the vector between the current iterate and the centre of each approximation $\hat{\boldsymbol{w}}_t^n - \boldsymbol{w}_t$. The second term is linear in $\boldsymbol{w} - \boldsymbol{w}_t$, for all linear approximations. This gives the following expanded form for the bundle as:

$$\max_{n\in[N]} \left\{\ell_{z_t^n}(\hat{\boldsymbol{w}}_t^n) - \nabla\ell_{z_t^n}(\hat{\boldsymbol{w}}_t^n)^\top(\hat{\boldsymbol{w}}_t^n - \boldsymbol{w}_t) + \nabla\ell_{z_t^n}(\hat{\boldsymbol{w}}_t^n)^\top(\boldsymbol{w} - \boldsymbol{w}_t)\right\}. \quad (4.2)$$

If we define $b_t^n = \ell_{z_t^n}(\hat{\boldsymbol{w}}_t^n) - \nabla\ell_{z_t^n}(\hat{\boldsymbol{w}}_t^n)^\top(\hat{\boldsymbol{w}}_t^n - \boldsymbol{w}_t)$, we can simplify the expression into the desired form. Hence, we now consider the BORAT proximal problem at time t with a bundle of size N as:

$$\boldsymbol{w}_{t+1} = \underset{\boldsymbol{w}\in\Omega}{\operatorname{argmin}} \left\{ \frac{1}{2\eta}\|\boldsymbol{w} - \boldsymbol{w}_t\|^2 + \max_{n\in[N]} \left\{ \nabla\ell_{z_t^n}(\hat{\boldsymbol{w}}_t^n)^\top(\boldsymbol{w} - \boldsymbol{w}_t) + b_t^n \right\} \right\}. \tag{4.3}$$

For BORAT we always set $\hat{\boldsymbol{w}}_t^1 = \boldsymbol{w}_t$ and use the last linear approximation to enforce the lower bound on the loss. This is done by setting $\nabla\ell_{z_t}(\hat{\boldsymbol{w}}_t^N) = [0, ..., 0]^\top$, $b_t^N = \mathrm{B} = 0$. We give details on how we select $\hat{\boldsymbol{w}}_t^n$ for $n \in \{2, ..., N-1\}$ in Section 4.3.4. Thus, each bundle is composed of $N-1$ linear approximations of the function, and the lower bound on the loss. These linear approximations of the loss need to be stored in memory during each step. Hence, in order to fit on a single GPU we only consider small bundle sizes in this work ($N \leq 5$). For clarity we depict a 1D example for a bundle with $N = 3$ in Figure 4.2.

Unlike SGD, the BORAT proximal problem (4.3) is not smooth and hence cannot be solved by simply setting the derivatives to zero. Instead we choose to solve each proximal problem in the dual. Thus we refer to the proximal problem (4.3) as the primal problem.

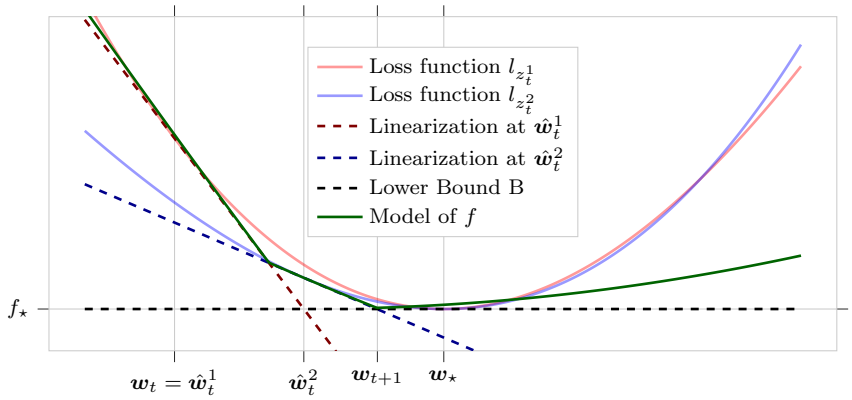


Figure 4.2: *Illustration of a BORAT bundle ($N = 3$) in 1D, shown in green. Two stochastic samples $\ell_{z_t^1}$ and $\ell_{z_t^2}$ of the loss function f are shown in red and blue (solid lines). The bundle is formed by taking the pointwise maximum of three linear approximations (dashed lines) and a proximal term. Two of these linear approximations are formed using the loss functions $\ell_{z_t^1}$ and $\ell_{z_t^2}$, and the last enforces the known lower bound on the loss. Here the BORAT approximation gives a more accurate model than the approximation used by ALI-G, which would only include the linearization at $\hat{\boldsymbol{w}}_t^1$ and the lower bound B. In this example this improved accuracy allows for BORAT to make more progress towards $\boldsymbol{w}_\star$. Specifically, BORAT would selected $\boldsymbol{w}_{t+1}$ for its next iterate, where as ALI-G would select $\hat{\boldsymbol{w}}_t^2$.*

4.3.3 Dual Problem

The dual of (4.3) is a constrained concave quadratic maximisation over N dual variables $\alpha^1, ..., \alpha^N$, and can be concisely written as follows (see Appendix A for derivation):

$$\boldsymbol{\alpha}_t = \underset{\boldsymbol{\alpha} \in \Delta_N}{\operatorname{argmax}} D(\boldsymbol{\alpha}), \text{ where } \quad D(\boldsymbol{\alpha}) = -\frac{\eta}{2} \boldsymbol{\alpha}^\top A_t^\top A_t \boldsymbol{\alpha} + \boldsymbol{\alpha}^\top \boldsymbol{b}_t. \tag{4.4}$$

Here A_t is a $N \times d$ matrix whose n^{th} row is $\nabla \ell_{z_t}(\hat{\boldsymbol{w}}_t^n)$. We define $\boldsymbol{b}_t = [b_t^1, ..., b_t^N]^\top$, $\boldsymbol{\alpha} = [\alpha^1, \alpha^2, \ldots, \alpha^N]^\top$ and Δ_N is a probability simplex over the N variables. The dual problem (4.4) has a number of features that make it more appealing for optimisation than the primal (4.3). First, the primal problem is defined over the parameter space $\boldsymbol{w} \in \mathbb{R}^d$, where d is in the order of thousands if not millions for modern deep neural networks. In contrast, the dual variables are of dimension N, where N is likely a small number due to the memory requirements. Second, the dual is smooth

and hence allows for faster convergence with standard optimisation techniques. Furthermore, as will be seen shortly, we use the fact that the dual feasible region is a tractable probability simplex to design a customised algorithm for its solution. We detail this algorithm in Section 4.3.5. Once we have found the dual solution α_t, we recover the following update from the KKT conditions:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta A_t \boldsymbol{\alpha}_t. \tag{4.5}$$

The form of the update (4.5) deserves some attention. Since each row of A_t is either the gradient of the loss $\ell_{z_t^n}$ or a zero vector, and α_t belongs to the probability simplex, the update step moves in the direction of a non-negative linear combination of negative gradients $-\nabla\ell_{z_t^n}(\hat{\boldsymbol{w}}_t^n)$. Due to the definitions of $\nabla\ell_{z_t}(\hat{\boldsymbol{w}}_t^N) = [0, ..., 0]^\top$ and $b_t^N = 0$, any weight given to α^N reduces the magnitude of the resulting step. This has the effect that as the loss value gets close to zero BORAT automatically decreases the size of the step taken.

4.3.4 Selecting Additional Linear Approximations for the Bundle

When constructing bundles of size $N > 2$, we are faced with two design decisions regarding how to select additional linear approximations to add to the bundle. First, where in parameter space should we construct the additional linear approximations $\hat{\boldsymbol{w}}_t^n$? And second, should we use the same mini-batch of data when constructing the stochastic linear approximations, or should we sample a new batch to evaluate each linear approximation?

Selecting $\hat{\boldsymbol{w}}_t^n$. Ideally, we would select the location of the linear approximations $\hat{\boldsymbol{w}}_t^n$ for $n \in \{2, ..., N-1\}$ in order to maximise the progress made towards $\boldsymbol{w}_\star$ at each step. However, without the knowledge of $\boldsymbol{w}_\star$ *a priori* and due to the high

dimensional and non-convex nature of problem ($\mathcal{P}$), this is infeasible. Instead, we make use of a heuristic. Inspired by the work of previous bundle methods for convex problems (Smola et al., 2007; Asi and Duchi, 2019) we select $\hat{\boldsymbol{w}}_t^n$ using the following method:

$$\hat{\boldsymbol{w}}_t^n = \underset{\boldsymbol{w}\in\mathbb{R}^d}{\operatorname{argmin}} \left\{ \frac{1}{2\eta}\|\boldsymbol{w} - \boldsymbol{w}_t\|^2 + \max_{k\in[n-1]} \left\{ \nabla \ell_{z_t^k}(\hat{\boldsymbol{w}}_t^k)^\top (\boldsymbol{w} - \boldsymbol{w}_t) + b_t^k \right\} \right\}. \tag{4.6}$$

In other words we construct the bundle by recursively adding linear approximations centred at the current optimum. This method of selecting additional linear approximations is appealing as it requires no extra hyperparameters and helps refine the approximation in the region of parameter space that would be explored by the next update.

Re-sampling z for additional linear approximations. When constructing additional linear approximations we choose to re-sample z. Concretely, we use a new mini-batch of data to construct each stochastic linear approximation. While it is possible to construct all $N-1$ non-zero linear approximations using the same batch of data we find this does not work well in practice. Indeed, such a method behaves similarly to taking multiple consecutive steps of SGD on the same mini-batch, which tends to produce poor optimisation performance.

Summary. We now summarise the bundle construction procedure for $N > 2$. We construct a bundle around $\boldsymbol{w}_t$ by first using two linear approximations, one centred at $\boldsymbol{w}_t$ and the second given by a known lower bound on the loss. We then sequentially add linear approximations until we have N. These extra linear approximations are constructed one at a time using new batches of data and centred around the point that is the current minimizer of the bundle. We note that each parameter update of BORAT uses $N-1$ batches of data. Therefore, BORAT updates the parameters $N-1$ fewer times than SGD in an epoch (given the same batch size). Once the

bundle is fully constructed we update $\boldsymbol{w}_t$. At this stage we apply momentum (if enabled) and project back on the feasible set Ω.

The construction of the bundle requires solving a minimization problem for each newly added piece when $N \geq 2$. Therefore, it is critical that such a problem gets solved very efficiently. We next detail how we do this by solving the corresponding dual problem for $N \geq 2$.

4.3.5 Efficient Dual Algorithm to Compute $N \geq 2$ Linear Pieces

In the general case ($N > 2$), the dual has more than 1 degrees of freedom and cannot be written as a 1D minimisation. Thus, the method derived for the case $N = 2$ is no longer applicable. This means it is not possible to obtain a simple closed form update. We must instead run an inner optimisation to solve (4.4) at each step. Many methods exist for maximising a concave quadratic objective over a simplex. Two algorithms particularly well suited to problems of the form (4.4) are Conditional Gradients (Frank and Wolfe, 1956) or Homotopy Methods (Bach et al., 2011). However, we propose a novel algorithm that exploits the fact that N is small to find the maximum directly. This method decomposes the problem of solving (4.4) into several subproblems, which provides two computational conveniences. First, the BORAT algorithm repeatedly searches for solutions to a bundle with only one newly added linear approximation since the last search. As one might expect, this task shares a great number of subproblems with the previous solution and allows for much of the computation to be reused. Second, our dual algorithm allows for a parallel implementation, which makes it very fast on the hardware commonly used for deep learning. To illustrate the efficiency of our dual method, the run time of the dual solution, that is finding $\boldsymbol{\alpha}_t$ once we have constructed (4.4), takes less than 5% of the time spent in the call of the optimiser. Note, due to the large size of the networks and the small size of N, the majority of the call time is dominated by the

computation of $A_t^\top A_t$ and $A_t \boldsymbol{\alpha}_t$.

We now formally introduce our dual solution algorithm. Our method uses the observation that at the optimal solution in a simplex the partial derivatives will be equal for all non-zero dimensions. This observation can be formally stated as:

Proposition 1 (Simplex Optimality Conditions)**.** *Let $F : \mathbb{R}^N \to \mathbb{R}$ be a concave function. Let us define $\boldsymbol{\alpha}_* = \operatorname{argmax}_{\boldsymbol{\alpha} \in \Delta} F(\boldsymbol{\alpha})$. Then there exists $c \in \mathbb{R}$ such that:*

$$\forall n \in [N] \text{ such that } \alpha_*^n > 0, \text{ we have: } \left.\frac{\partial F(\boldsymbol{\alpha})}{\partial \alpha^n}\right|_{\boldsymbol{\alpha}=\boldsymbol{\alpha}_*} = c. \tag{4.7}$$

In other words, the value of the partial derivative is shared among all coordinates of $\boldsymbol{\alpha}_*$ that are non-zero.

This proposition can be easily proved by contradiction. If the partial derivatives are not equal, then moving in the direction of the largest would result in an increase in function value. Likewise, moving in the negative direction would produce a decrease. Hence, the current point cannot be optimal. Please see Appendix C for a formal proof of Proposition 1.

In the following paragraphs we explain how this proposition can be used to break up the task of solving problem (4.4) into $2^N - 1$ subproblems. These subproblems arise from considering a unique subset I of non-zero dimensions of $\boldsymbol{\alpha}$. Each of these subproblems involves finding the unconstrained optimum and checking if this point lies within the simplex. We now give an example of a single subproblem. To simplify notation, let $Q \triangleq \eta A_t^\top A_t$. We note that:

$$\left.\frac{\partial D(\boldsymbol{\alpha})}{\partial \boldsymbol{\alpha}}\right|_{\boldsymbol{\alpha}=\boldsymbol{\alpha}_*} = -Q\boldsymbol{\alpha}_* + \boldsymbol{b}_t. \tag{4.8}$$

If we knew that $\boldsymbol{\alpha}_*$ had exclusively non-zero coordinates, then by applying Proposition (1) to the dual objective D we can recover a solution $\boldsymbol{\alpha}_*$ by solving the following

linear system:

$$\begin{bmatrix} \boldsymbol{\alpha}_* \\ -c \end{bmatrix} = \texttt{solve}_{\boldsymbol{x} \geq 0} \left(\begin{bmatrix} Q & \mathbf{1} \\ \mathbf{1}^\top & 0 \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} \boldsymbol{b}_t \\ 1 \end{bmatrix} \right). \tag{4.9}$$

The first N rows of the system would enforce that $\boldsymbol{\alpha}_*$ satisfies the condition given by Proposition 1, and the last row of the linear system would ensure that the coordinates of $\boldsymbol{\alpha}_*$ sum to one. In the general case, we do not know which coordinates of $\boldsymbol{\alpha}_*$ are non-zero. However, since typical problems are in low-dimension N, we can enumerate all possibilities of subsets of non-zero coordinates for $\boldsymbol{\alpha}_*$.

We detail this further. We consider a non-empty subset $I \subseteq [N]$, for which we define:

$$Q_{[I \times I]} \triangleq (Q_{i,j})_{i \in I, j \in I}\,, \quad \boldsymbol{b}_{[I]} \triangleq (b_{t,i})_{i \in I}\,. \tag{4.10}$$

We then solve the corresponding linear subsystem:

$$\begin{bmatrix} \boldsymbol{\phi}^{(I)} \\ -c \end{bmatrix} \triangleq \texttt{solve}_{\boldsymbol{x} \geq 0} \left(\begin{bmatrix} Q_{[I \times I]} & \mathbf{1} \\ \mathbf{1}^\top & 0 \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} \boldsymbol{b}_{[I]} \\ 1 \end{bmatrix} \right). \tag{4.11}$$

This $\boldsymbol{\phi}^{(I)} \in \mathbb{R}^{|I|}$ can then be lifted to $\mathbb{R}^N$ by setting zeros at coordinates not contained in I. Formally, we define $\boldsymbol{\psi}^{(I)} \in \mathbb{R}^N$ such that:

$$\forall i \in [N],\ \psi_i^{(I)} = \begin{cases} \phi_i^{(I)} \text{ if } i \in I, \\ 0 \text{ otherwise.} \end{cases} \tag{4.12}$$

Therefore, given $I \subseteq [N]$, we can generate a candidate solution $\boldsymbol{\psi}^{(I)}$ for problem (4.4) by solving a linear system in dimension $|I|$. In the following proposition, we establish that doing so for all possibilities of I guarantees finding the correct solution:

Proposition 2 (Problem Equivalence)**.** *We define the set of feasible solutions reached by the different candidates $\boldsymbol{\psi}^{(I)}$:*

$$\Psi = \left\{ \boldsymbol{\psi}^{(I)} : I \subseteq [N], I \neq \varnothing \right\} \cap \Delta_N. \tag{4.13}$$

Then we have that:

$$\underset{\boldsymbol{\alpha}\in\Delta_N}{\operatorname{argmax}}\, D(\boldsymbol{\alpha}) = \underset{\boldsymbol{\psi}\in\Psi}{\operatorname{argmax}}\, D(\boldsymbol{\psi}). \tag{4.14}$$

In other words, we can find the optimal solution of (4.4) by enumerating the members of Ψ *and picking the one with highest objective value.*

This proposition is trivially true because Ψ is simply the intersection between (i) the original feasible set Δ_N and (ii) the set of vectors that satisfy the necessary condition of Optimality given by Proposition 1. This insight results in Algorithm 1 which returns an optimal solution to the dual problem. We characterise this claim in the following proposition.

Proposition 3 (Sets of Solutions)**.** *Algorithm 1 returns a solution* $\boldsymbol{\alpha}^*$ *that satisfies* $\boldsymbol{\alpha}^* \in \operatorname{argmax}_{\boldsymbol{\alpha}\in\Delta_N} D(\boldsymbol{\alpha})$*. This is true even when the dual does not have a unique solution. Proof given in Appendix A.3.*

Algorithm 1 Dual Optimisation Algorithm

Require: η, N, $\mathcal{P} = \{S : S \subseteq \{1, 2, ..., N\}, S \neq \varnothing\}$, $Q = \eta A_t^\top A_t$, $\boldsymbol{b}_t$, $d_{max} = 0$.

1: **for** $I \in P$ **do**

2: $\quad \hat{Q} = \begin{bmatrix} Q_{[I\times I]} & \mathbf{1} \\ \mathbf{1}^\top & 0 \end{bmatrix}$, $\hat{\boldsymbol{b}} = \begin{bmatrix} \boldsymbol{b}_{[I]} \\ 1 \end{bmatrix}$ ▷ see Equation (4.10) for definitions

3: $\quad \boldsymbol{\phi}^{[I]} = \texttt{solve}_{\boldsymbol{x}}\left(\hat{Q}\boldsymbol{x} = \hat{\boldsymbol{b}}\right)$ ▷ solve the subsystem, see Equation (4.11)

4: $\quad$ **if** $\boldsymbol{\phi}_i^{[I]} \geq 0,\ \forall i \in \{1, 2, ..., |I|\}$ **then** ▷ check for non-negativity of solution

5: $\quad\quad \boldsymbol{\psi}^{(I)} = \texttt{select}(\boldsymbol{\phi}^{[I]}, I)$ ▷ select elements according to Equation (4.12)

6: $\quad\quad d = -\frac{1}{2}\boldsymbol{\psi}^{(I)\top} Q \boldsymbol{\psi}^{(I)} + \boldsymbol{\psi}^{(I)\top}\boldsymbol{b}_t$ ▷ compute the dual value

7: $\quad\quad$ **if** $d \geq d_{max}$ **then** ▷ save maximum value

8: $\quad\quad\quad d_{max} = d, \boldsymbol{\alpha}^* = \boldsymbol{\psi}^{(I)}$

9: **Return** $\boldsymbol{\alpha}^*$ ▷ return optimal value

Procedurally, Algorithm 1 starts by computing $Q = -\eta A_t^\top A_t$ and $\boldsymbol{b}_t$ to form a "master" system $Q\boldsymbol{x} = \boldsymbol{b}$. We consider each of the $2^N - 1$ subsystems of $Q\boldsymbol{x} = \boldsymbol{b}_t$ defined by an element of the set I (lines 1-3), where I represents the set of non-zero dimensions of each subsystem. For each subsystem we get an independent subproblem. To ensure the solution to each subproblem will satisfy $\sum_{n=1}^{N} x^n = 1$ and all partial derivatives have equal value, an extra row-column is introduced to

each system (line 2). We then compute the point which satisfies the optimality conditions detailed in Proposition 1 for each subsystem, by solving for $\boldsymbol{x}$. We then check if each of these points is feasible, that is, belongs to Δ_N by examining signs $x^n \geq 0, \forall n \in \{1, ..., n\}$ (line 4). Note, we have by construction $\sum_{n=1}^{N} x^n = 1$. Finally, we select $\boldsymbol{\alpha}^*$ as the feasible point with maximum dual value (lines 7-8). The optimal $\boldsymbol{\alpha}^*$ is then used to define the weight update (4.5). For example, if $\boldsymbol{\alpha}^* = [1, 0, ..., 0]^\top$ an SGD step $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \nabla \ell_{z_t}(\boldsymbol{w}_t)$ will be taken.

The BORAT algorithm can be viewed as automatically picking the best step out of a maximum of $(2^N - 1)$ possible options, where each option has a closed form solution. Although the computational complexity of this method is exponential in N, this algorithm is still very efficient in practice for two reasons. First, we only consider small N. Second, as subproblems can be solved independently, it permits a parallel implementation. With a fully parallel implementation the time complexity of this algorithm reduces to $\mathcal{O}(N^3)$. Empirically, with such an implementation, for $N \leq 10$ we observe an approximately linear relationship between N and time taken per training epoch. See Appendix B.1 for a comparison of training epoch time between BORAT and SGD.

4.3.6 Computational Considerations

The method of adding linear approximations detailed in (4.6) requires running Algorithm 1 at each inner loop iteration. However in practice if we keep track of the best dual value when adding an additional element α^n one need only compute the 2^{n-1} new subproblems that include non-zero α^n. Thus, we actually only need to run Algorithm 1 fully once for each bundle, that is once per $N - 1$ batches.

4.3.7 Summary of the Algorithm

The full BORAT method is outlined in Algorithm 2. The bundle is constructed in lines 3-7. sequentially finding the minimiser and adding a linear approximation at

its location. Note we can reuse much of the computation between subsequent calls to Algorithm 1 line 6. In practice we effectively interleave a single call to Algorithm 1 for dual of dimension N with a single inner loop of Algorithm 2 lines 4-7. However we have shown them separately for clarity.

Once we have a bundle of size N we instead update $\boldsymbol{w}_t$, apply momentum and project back onto the feasible region Ω in lines 8-10.

In the majority of our experiments, we accelerate BORAT with Nesterov momentum see Algorithm 2.

We use Nesterov momentum as we find this helps produce strong generalisation performance. When momentum is not use we simply replace line 9 10 of Algorithm 2 with $\boldsymbol{w}_{t+1}, \hat{\boldsymbol{w}}_{t+1}^1 = \Pi_\Omega\left(\hat{\boldsymbol{w}}_t^{n+1}\right)$.

Algorithm 2 The BORAT Algorithm

Require: learning rate η, maximum bundle size $N \geq 2$, initial feasible $\boldsymbol{w}_0 \in \Omega$.

1: $t = 0$, $\hat{\boldsymbol{w}}_0^1 = \boldsymbol{w}_0$
2: **while** not converged **do**
3: **for** $n = 1, ..., N-1$ **do** ▷ add element to bundle
4: Sample $z_{t,n} \in \mathcal{Z}$, and compute $\ell_{z_t^n}(\hat{\boldsymbol{w}}_t^n)$, $\nabla \ell_{z_t^n}(\hat{\boldsymbol{w}}_t^n)$
5: compute $-\eta A_t^\top A_t$ and $\boldsymbol{b}_t$ ▷ add extra row-column
6: $\boldsymbol{\alpha}^* = \text{argmax}_{\boldsymbol{\alpha} \in \Delta_n} \left\{-\frac{\eta}{2}\boldsymbol{\alpha}^\top A_t^\top A_t \boldsymbol{\alpha} + \boldsymbol{\alpha}^\top \boldsymbol{b}_t\right\}$ ▷ see Algorithm 1 for details
7: $\hat{\boldsymbol{w}}_t^{n+1} = \boldsymbol{w}_t - \eta A_t \boldsymbol{\alpha}^*$
8: $\boldsymbol{v}_t = \mu \boldsymbol{v}_{t-1} - \eta A_t \boldsymbol{\alpha}^*$ ▷ nesterov Momentum
9: $\boldsymbol{w}_{t+1}, \hat{\boldsymbol{w}}_{t+1}^1 = \Pi_\Omega\left(\boldsymbol{w}_t + \mu \boldsymbol{v}_t\right)$ ▷ here Π_Ω is the projection onto Ω
10: $t = t + 1$
11: **end while**

4.4 Justification and Analysis

The interpolation setting gives, $f_\star = 0$ by definition. However, more subtly, it also allows the updates to rely on the stochastic estimate $\ell_{z_t}(\boldsymbol{w}_t)$ rather than the exact but expensive $f(\boldsymbol{w}_t)$. Intuitively, this is possible because in the interpolation setting, we know the global minimum is achieved for each loss function $\ell_{z_t}(\boldsymbol{w}_t)$ simultaneously. The following results formalise the convergence guarantee of BORAT in the

stochastic setting. Note, here we prove these results for BORAT with a minor modification, that is, all linear approximations are formed using the same mini-batch of data, $\ell_{z_t^n} = \ell_{z_t}$ for all $n \in \{2, ..., N-1\}$. In practice, we find that by using the same batch for constructing the $N-1$ linear approximations performs significantly worse than re-sampling a new mini-batch for each. We speculate this decrease in performance mirrors the decrease seen when running multiple SGD updates on the same batch before re-sampling. Specifically, the optimiser focuses too much on minimising ℓ_z on each step rather than the average f. However, this modification simplifies the proof significantly as we only require computing expectations with respect to a single mini-batch z_t.

First, we consider the standard convex setting, where we additionally assume the interpolation assumption is satisfied and that each ℓ_z is Lipschitz continuous. Next, we consider an important class of non-convex functions used for analysis in previous works related to interpolation (Vaswani et al., 2019).

Theorem 1 (Convex and Lipschitz)**.** *Let Ω be a convex set. We assume that for every $z \in \mathcal{Z}$, ℓ_z is convex and C-Lipschitz. Let $\boldsymbol{w}_\star$ be a solution of $(\mathcal{P})$, and assume that we have perfect interpolation: $\forall z \in \mathcal{Z}$, $\ell_z(\boldsymbol{w}_\star) = 0$. Then BORAT for $N \geq 2$ applied to f satisfies:*

$$f\left(\tfrac{1}{T+1}\sum_{t=0}^{T}\boldsymbol{w}_t\right) - f_\star \leq C\sqrt{\frac{\|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2}{(T+1)} + \frac{\|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2}{\eta(T+1)}}. \tag{4.15}$$

Hence, BORAT recovers the same asymptotic rate as SGD without the need to reduce the learning rate η during training. In the Appendix A.4 we show that for convex and β-smooth and the α-strongly convex settings BORAT recovers rates of $O(1/T)$ and $O(\exp(\alpha T))$, respectively.

We follow earlier work (Vaswani et al., 2019) and provide a convergence rate for BORAT applied to non-convex functions that satisfy the Restricted Secant Inequality (RSI). A function is said to satisfy the RSI condition with constant μ over the

set Ω if the following holds:

$$\forall \boldsymbol{w} \in \Omega, \langle \nabla \ell_z(\boldsymbol{w}), \boldsymbol{w} - \boldsymbol{w}_\star \rangle \leq \mu ||\boldsymbol{w} - \boldsymbol{w}_\star||. \tag{4.16}$$

Theorem 2 (RSI)**.** *We consider problems of type ($\mathcal{P}$). We assume ℓ_z satisfies the RSI with constant μ, smoothness constant β and perfect interpolation e.g. $l_z(\boldsymbol{w}^*) = 0$, $\forall z \in \mathcal{Z}$. Then if set $\eta \leq \hat{\eta} = \min\{\frac{1}{4\beta}, \frac{1}{4\mu}, \frac{\mu}{2\beta^2}\}$ then in the worst case we have:*

$$f(\boldsymbol{w}_{T+1}) - f^* \leq exp\left(\left(-\frac{3}{8}\hat{\eta}\mu\right)T\right)||\boldsymbol{w}_0 - \boldsymbol{w}^*||^2. \tag{4.17}$$

Thus in this setting BORAT recovers the same asymptotic rate as SGD.

4.5 Experiments

We split the experimental results into two sections. Section 4.5.1 demonstrates the strong generalisation performance of BORAT on a wide range of tasks. Here we compare BORAT to other single hyperparameter optimisation algorithms. Section 4.5.2 shows results for tasks where SGD and ALI-G are sensitive to the learning rate. For these tasks BORAT increases both the robustness to the learning rate and the task regularisation hyperparameter.

We choose to investigate BORAT with ($N = 3$) and ($N = 5$), and refer to the resulting algorithms as BORAT3 and BORAT5, respectively. For $N \geq 2$ BORAT uses more than one batch of data for each update. In order to give a fair comparison we keep the number of passes through the data and total gradient evaluations constant for all experiments. This has the effect that BORAT3 and BORAT5 respectively make a half and a quarter of the weight updates of SGD or ALI-G. The time per epoch of BORAT is very similar to those of all other methods, see Appendix C for more details. Hence, all methods have approximately the same run time per epoch. Consequently, faster convergence in terms of number of epochs translates into faster

convergence in terms of wall-clock time.

The code to reproduce our results is publicly available[1]. For baselines we use the official implementation in `PyTorch`, where available (Paszke et al., 2017). We use our implementation of L_4, which we unit-test against the official `TensorFlow` implementation (Abadi et al., 2015). We employ the official implementation of DFW[2] and we reuse their code for the experiments on SNLI and CIFAR.

4.5.1 Effectiveness of BORAT

We empirically compare BORAT to commonly used optimisation algorithms for deep learning using standard loss functions. In this section the hyperparameters of each algorithm are thoroughly cross-validated to select a best performing model. We consider a wide range of problems: training a wide residual network on SVHN, training a Bi-LSTM on the Stanford Natural Language Inference data set, training both wide residual networks and densely connected networks on the CIFAR data sets, and lastly, training a wide residual network on the Tiny ImageNet data set. For these problems, we demonstrate that BORAT3 and BORAT5 obtain comparable performance to SGD with a hand-tuned learning rate schedule, and typically outperform adaptive gradient methods. Finally, in order to demonstrate the scalability of BORAT to large-scale settings, we empirically assess the performance of BORAT and its competitors in terms of training objective on CIFAR-100 and ImageNet. Note that the tasks of training wide residual networks on SVHN and CIFAR-100 are part of the DeepOBS benchmark (Schneider et al., 2019), which aims to standardise baselines for deep learning optimisers. In particular, these tasks are among the most difficult ones of the benchmark, because the SGD baseline benefits from a manual schedule for the learning rate whereas BORAT uses a single fixed value. Despite this, our set of experiments demonstrates that BORAT obtains competitive performance in relation to SGD. In addition, our method significantly

[1] https://github.com/oval-group/borat
[2] https://github.com/oval-group/dfw

outperforms adaptive gradient methods. All experiments were performed on a single GPU (SVHN, SNLI, CIFAR) or on up to 4 GPUs (ImageNet).

4.5.1.1 Wide Residual Networks on SVHN

Setting. The SVHN data set contains 73k training samples, 26k testing samples and 531k additional easier samples. From the 73k difficult training examples, we select 6k samples for validation; we use all remaining (both difficult and easy) examples for training, for a total of 598k samples. We train a wide residual network 16-4 following Zagoruyko and Komodakis (2016).

Method. For SGD, we use the manual schedule for the learning rate of Zagoruyko and Komodakis (2016). For L_4Adam and L_4Mom, we cross-validate the main learning rate hyperparameter α to be in $\{0.0015, 0.015, 0.15\}$ (0.15 is the value recommended by Rolinek and Martius (2018)). For other methods, the learning rate hyperparameter is tuned as a power of 10. The ℓ_2 regularisation is cross-validated in $\{0.0001, 0.0005\}$ for all methods except ALI-G and BORAT. For ALI-G and BORAT, the regularisation is expressed as a constraint on the ℓ_2-norm of the parameters, and its maximal value is set to 100. SGD, ALI-G, BORAT and BPGrad use a Nesterov momentum of 0.9. All methods use a dropout rate of 0.4 and a fixed budget of 160 epochs, following Zagoruyko and Komodakis (2016). A batch size of 128 is used for all experiments.

Results. A comparison with other methods is presented in Table 4.1. On this relatively easy task, most methods achieve about 98% test accuracy. Despite the cross-validation, L_4Mom does not converge on this task. However, note that L_4Adam achieves accurate results. Even though SGD benefits from a hand-designed schedule, BORAT and other adaptive methods obtain comparable performance on this task.

Test Accuracy on SVHN (%)			
Adagrad	98.0	BPGrad	98.1
AMSGrad	97.9	L_4Adam	**98.2**
DFW	98.1	ALI-G	98.1
L_4Mom	19.6	BORAT3	98.1
Adam	97.9	BORAT5	98.1
SGD	98.3	SGD†	98.4

Table 4.1: *In red, SGD benefits from a hand-designed schedule for its learning rate. In black, adaptive methods, including ALI-G, have a single hyperparameter for their learning rate. SGD† refers to the performance reported by Zagoruyko and Komodakis (2016).*

4.5.1.2 Bi-LSTM on SNLI

Setting. We train a Bi-LSTM of 47M parameters on the Stanford Natural Language Inference (SNLI) data set (Bowman et al., 2015). The SNLI data set consists of 570k pairs of sentences, with each pair labelled as entailment, neutral or contradiction. This data set is used as a pre-training corpus for transfer learning to many other natural language tasks where labelled data is scarcer (Conneau et al., 2017), much like ImageNet is used for pre-training in computer vision. We follow the protocol of Berrada et al. (2019) and we reuse their results for the baselines.

Method. For L_4Adam and L_4Mom, the main hyperparameter α is cross-validated in $\{0.015, 0.15\}$ – compared to the recommended value of 0.15, this helped convergence and considerably improved performance. The SGD algorithm benefits from a hand-designed schedule, where the learning rate is decreased by 5 when the validation accuracy does not improve. Other methods use adaptive learning rates and do not require such a schedule. Thus, the value of the main hyperparameter η is cross-validated as a power of ten for the other methods. Following the implementation by Conneau et al. (2017), no ℓ_2 regularisation is used. The algorithms are evaluated with the Cross-Entropy (CE) loss and the multi-class hinge loss (SVM), except for DFW which is designed for use with an SVM loss only. For all optimisation

algorithms, the model is trained for 20 epochs, using a batch size of 64, following Conneau et al. (2017).

Test Accuracy on SNLI (%)					
	CE	SVM		CE	SVM
Adagrad*	83.8	84.6	Adam*	84.5	85.0
AMSGrad*	84.2	85.1	BPGrad*	83.6	84.2
DFW*	-	**85.2**	L_4Adam	83.3	82.5
L_4Mom	83.7	83.2	BORAT3	84.4	**85.2**
ALI-G	**84.8**	**85.2**	BORAT5	84.5	**85.2**
SGD*	84.7	85.2	SGD†	84.5	-

Table 4.2: *In red, SGD benefits from a hand-designed schedule for its learning rate. In black, adaptive methods have a single hyperparameter for their learning rate. With an SVM loss, DFW and ALI-G are procedurally identical algorithms – but in contrast to DFW, ALI-G can also employ the CE loss. Methods in the format X* reuse results from Berrada et al. (2019). SGD† is the result from Conneau et al. (2017).*

Results. Table 4.2 compares BORAT against the other optimisers. When in combination with the hinge loss BORAT achieves the joint best performance with ALI-G, DFW and SGD with a learning rate schedule. When considering the CE loss ALI-G performed best by a small margin.

4.5.1.3 Wide Residual Networks and Densely Connected Networks on CIFAR

Setting. We follow the methodology of Berrada et al. (2019). We test two architectures: a Wide Residual Network (WRN) 40-4 (Zagoruyko and Komodakis, 2016) and a bottleneck DenseNet (DN) 40-40 (Huang et al., 2017). We use 45k samples for training and 5k for validation. The images are centred and normalized per channel. We apply random horizontal flipping and random crops for data augmentation.

Method. We compare BORAT to ALI-G and other common single hyperparameter optimisers. Here, we cross validate the hyperparameters in order to find the

best performing network for each method. AMSGrad was selected in Berrada et al. (2019) because it was the best adaptive method on similar tasks, outperforming in particular Adam and Adagrad. In addition to these baselines, we also provide the performance of L_4Adam, L_4Mom, AdamW (Loshchilov and Hutter, 2019) and Yogi (Zaheer et al., 2018). We follow the cross validation scheme of Berrada et al. (2019) restating it here for completeness. All methods employ the CE loss only, except for the DFW algorithm, which is designed to use an SVM loss. The batch size is cross-validated in $\{64, 128, 256\}$ for the DN architecture, and $\{128, 256, 512\}$ for the WRN architecture. For L_4Adam and L_4Mom, the learning rate hyperparameter α is cross-validated in $\{0.015, 0.15\}$. For AMSGrad, AdamW, Yogi, DFW, ALI-G, and BORAT the learning rate hyperparameter η is cross-validated as a power of 10 (in practice $\eta \in \{0.1, 1\}$ for BORAT). SGD, DFW, and BORAT use a Nesterov momentum of 0.9. For all methods excluding ALI-G, BORAT, and AdamW, the ℓ_2 regularisation is cross-validated in $\{0.0001, 0.0005\}$ on the WRN architecture, and is set to 0.0001 for the DN architecture. For AdamW, the weight-decay is cross-validated as a power of 10. For ALI-G, and BORAT, ℓ_2 regularisation is expressed as a constraint on the norm of the vector of parameters; and its value is cross-validated in $\{50, 75, 100\}$. For all optimisation algorithms, the WRN model is trained for 200 epochs and the DN model for 300 epochs, following Zagoruyko and Komodakis (2016) and Huang et al. (2017) respectively.

Results. Table 4.3 details the results of the comparison of BORAT against other single hyperparameter optimisers for the CE loss only. In this setting, BORAT outperformed AMSGrad, AdamW, Yogi, L_4Adam and L_4 Mom, and constant step size SGD by a large margin. ALI-G achieved ever so slightly better results than BORAT on CIFAR-100 using the WRN model, and performed about equally or slightly worse on the other tasks. The next best method was DFW which also has the restriction that it can only be used in conjunction with the hinge loss. ALI-G and

BORAT produced the test accuracy achievable using SGD with the manual learning rate schedules from Huang et al. (2017) and Zagoruyko and Komodakis (2016) for half of the model data set combinations considered. For these tasks BORAT provided state-of-the-art results without the need for the learning rate to be manually adapted throughout training. For the remaining two combinations, training a DN on CIFAR-10 and training a WRN on CIFAR-100, BORAT lags in performance by approximately 0.2% and 2%, respectively, with minor variation depending on which version of BORAT is used. Note, SGD with a hand-tuned learning rate schedule provides a reasonable upper limit on the generalisation performance achievable, due to the amount of time researchers have put into improving these schedules.

Test Accuracy on CIFAR (%)				
	CIFAR-10		CIFAR-100	
	WRN	DN	WRN	DN
SGD	91.2	91.5	67.8	67.5
AMSGrad	90.8	91.7	68.7	69.4
AdamW	92.1	92.6	69.6	69.5
Yogi	91.2	92.1	68.7	69.6
DFW	94.2	94.6	76.0	73.2
L_4Adam	90.5	90.8	61.7	60.5
L_4Mom	91.6	91.9	61.4	62.6
ALI-G	**95.4**	94.5	**76.1**	76.2
BORAT3	**95.4**	**94.9**	76.0	**76.5**
BORAT5	95.0	**94.9**	75.8	75.7

Table 4.3: *Test accuracy of single hyperparameter optimisation methods. Each reported result is an average over three independent runs; the standard deviations and optimal hyperparameters are reported in Appendix B.3 (the standard deviations are at most* 0.4 *for ALI-G and BORAT).*

4.5.1.4 Wide Residual Networks on Tiny ImageNet

Setting. Tiny ImageNet contains 200 classes for training where each class has 500 images. The validation set contains 10,000 images. All images are RGB with 64x64 pixels. Thus, Tiny ImageNet is significantly more challenging than the CIFAR data sets. The test-set contains 10,000 images, however the ground truth labels are

not freely available. We again use the Wide Residual Network (WRN) detailed in Section 4.5.1.1. The images are centred and normalized per channel. We apply standard data augmentation with random horizontal flipping and random crops.

Method. We investigate the ability of SGD (with a constant step size), ALI-G, and BORAT to train a WRN on Tiny ImageNet. We use Adam (Kingma and Ba, 2015) as an indicator of what can be expected from a popular adaptive method applied to the same task. We make use of both the cross entropy (CE) and multi-class hinge (SVM) losses. The learning rate hyperparameter η is cross-validated in powers of 10, a batch size of 128 and a training time of 200 epochs was used for all experiments. The ℓ_2 regularisation is cross-validated in powers of 10 for Adam. For constant step size SGD, ALI-G, and BORAT we make use of the constrained base regularisation detailed in Section 3.9.1, cross-validating $r \in \{50, 100, 150, 200, 250\}$. Additionally, all methods excluding Adam use a Nesterov momentum of 0.9.

Results. The best results for SGD, ALI-G, BORAT, and Adam are shown in Table 4.4. For both losses Adam performs worst, only achieving 2.4% validation accuracy when optimising the SVM loss. The best results are achieved by BORAT with $N = 5$ and $N = 3$ for the CE and SVM losses respectively. These results suggest the added gain of BORAT is more pronounced for larger more challenging data sets.

Validation Accuracy on Tiny ImageNet (%)		
	CE Loss	Hinge Loss
Adam	55.0	2.4
SGD	59.4	23.2
ALI-G	61.1	24.9
BORAT3	61.1	**44.1**
BORAT5	**62.1**	39.4

Table 4.4: *Validation accuracy of single hyperparameter optimisation methods on the Tiny ImageNet data set for cross entropy and hinge loss.*

4.5.1.5 Comparing Training Performance on CIFAR-100

In this section we empirically assess the performance of BORAT and its competitors in terms of training objective on CIFAR-100. In order to have comparable objective functions, the ℓ_2 regularisation is deactivated. The learning rate is selected as a power of ten for best final objective value, and the batch size is set to its default value. For clarity, we only display the performance of SGD, Adam, Adagrad, and BORAT (DFW does not support the cross-entropy loss). The L_4 methods diverge in this setting. Here, SGD uses a constant learning rate to demonstrate the need for adaptivity. Therefore, all methods use one hyperparameter for their learning rate. All methods use a fixed budget of 200 epochs for WRN-CIFAR-100 and 300 epochs for DN-CIFAR-100. As can be seen in Figure 4.3, BORAT provides similar training performance to ALI-G and outperforms the other baseline algorithms on both tasks. Here, BORAT3 and BORAT5 are slightly slower than ALI-G due to the lower number of parameter updates performed.

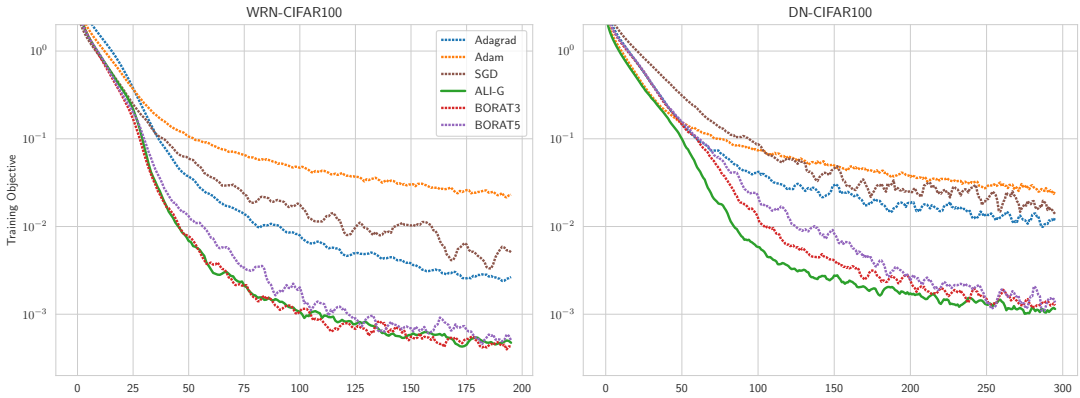


Figure 4.3: *Objective function over the epochs on CIFAR-100 (smoothed with a moving average over 5 epochs). ALI-G and BORAT reach a value that is an order of magnitude better than the baselines.*

4.5.1.6 Training at Large Scale

We demonstrate the scalability of BORAT by training a ResNet18 (He et al., 2016) on the ImageNet data set and comparing against ALI-G. In order to satisfy the

interpolation assumption, we employ a loss function tailored for top-5 classification (Lapin et al., 2016), and we do not use data augmentation. Our focus here is on the training objective and accuracy. ALI-G and BORAT use the following training setup: a batch size of 1024 split over 4 GPUs, an ℓ_2 maximal norm of 400 for $\boldsymbol{w}$, a maximal learning rate of 10, and no momentum. As can be seen in Figure 4.4, ALI-G and BORAT reach 99% top-5 accuracy in 12 epochs, and accurately minimise the objective function to $2 \cdot 10^{-4}$ within 90 epochs.

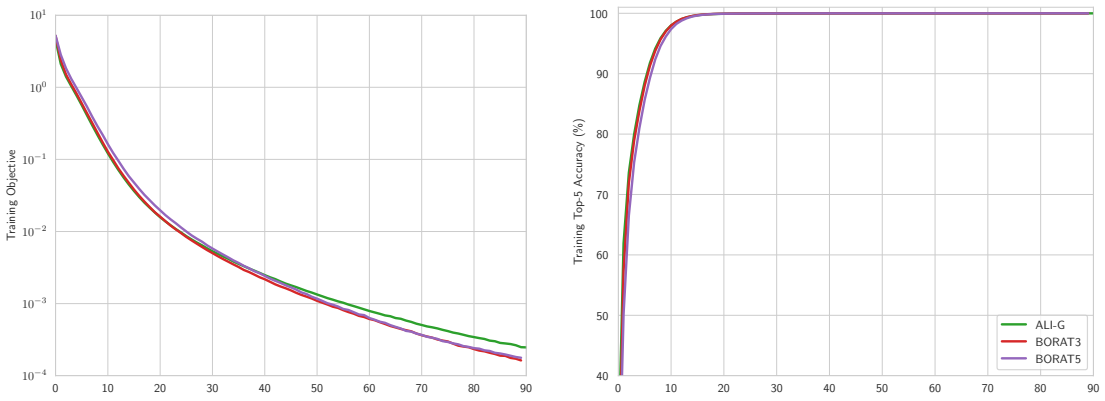


Figure 4.4: *Training performance of a ResNet-18 learning with different versions of BORAT on ImageNet. Note that all versions converge at similar rates even though BORAT3 and BORAT5 make far fewer updates for larger values of N.*

4.5.2 Robustness of BORAT

In this section we show that using additional linear approximations increases the robustness of BORAT to its learning rate η and the problem regularisation amount r. BORAT produces high accuracy models for a wider range of values than SGD and ALI-G. Hence, on problems where SGD and ALI-G are sensitive to their learning rate BORAT with $N \geq 2$ produces the best performance. These results demonstrate the advantage of using a larger bundle to model the loss in each proximal update.

In order to illustrate this increased robustness we assessed the stability of constant step size SGD, ALI-G, BORAT3, and BORAT5 to their hyperparameters. We additionally provide results for Adam in Appendix B.2. Robustness is assessed by

completing a grid search over η and r for a number of tasks while holding the batch size and epoch budget constant. This allows us to assess the range of values where these algorithms produce high accuracy models. We perform this grid search for six tasks split over the CIFAR-100 and Tiny ImageNet data sets. We choose these data sets as they are challenging yet a model capable of interpolation can still fit on a single GPU. We compare against SGD with a constant learning rate for two reasons. First, SGD effectively uses a bundle of size 1, composed of only the linearization around the current iterate. Second, constant step size SGD has a single learning rate hyperparameter with the same scale as BORAT and also permits the constraint-based regularisation described in Section 3.9.1.

4.5.2.1 Wide Residual Networks on CIFAR-100 and Tiny ImageNet

Setting. The CIFAR-100 and Tiny ImageNet data sets are described in detail in Sections 4.5.1.3 and 4.5.1.4, respectively.

Method. We ran four separate experiments on the CIFAR-100 data set. The first two of these experiments examine the robustness to hyperparameters of SGD, ALI-G, and BORAT when in combination with the cross entropy (CE) and multi-class hinge (SVM) losses. For the second two experiments on CIFAR-100 we investigate the same algorithms in the presence of label noise. We limit ourselves to the CE loss for these two experiments. Label noise is applied by switching the label of the images in the training set with probability p to a random class label in the same superclass. We repeat the experiments without label noise on the more challenging Tiny ImageNet data set, resulting in six different settings. For all algorithms we train a wide residual network (WRN) detailed in Section 4.5.1.3 and make use of a Nesterov momentum of 0.9 as we found its use produced superior generalisation performance. For all experiments we use a batch size of 128 and perform a grid search over the 20 hyperparameter combinations given by the Cartesian product of

$r \in \{50, 100, 150, 200, 250\}$ and $\eta \in \{0.01, 0.1, 1.0, 10.0\}$.

Figure 4.5: *Comparison of SGD, ALI-G and BORAT's robustness to hyperparameters when increasing N for the CE and SVM losses. Experiments are performed on the CIFAR-100 data set. Colour represents test performance, where darker colours correspond to higher values. For both losses, increasing N allows for higher learning rates to be used while still producing convergent behaviour. Additionally, for the CE loss and $\eta \in \{1.0, 10.0\}$ larger N allows for a smaller r or greater levels of regularisation to be used. Consequently, increasing N improves the overall robustness to hyperparameters, while not sacrificing generalisation performance.*

Results. We first discuss the results for the CIFAR-100 data set. Figure 4.5 details the robustness of SGD, ALI-G and BORAT for the CE and SVM losses without label noise. We provide results for the Adam optimiser in Appendix B.2. In these two settings ALI-G exhibits similar robustness to constant step size SGD, however, it outperforms SGD in terms of the performance of the best model trained. On the CE loss, SGD and ALI-G are reasonably robust, providing good results for the majority of hyperparameter combinations with $\eta \leq 1$. For the SVM loss all algorithms are sensitive to their learning rate η. SGD and ALI-G only produce an increase in accuracy for small values of η, hence no model achieves high accuracy in the 200 epoch budget. For both losses, increasing N improves the robustness

and produces convergent behaviour for larger values of η and smaller values of r. This is particularly pronounced for the SVM loss. Here, permitting larger η allows for a high accuracy network to be trained within the 200 epoch budget. For both losses $\eta = 0.01$ BORAT3 and BORAT5 slightly underperformed compared to ALI-G and SGD. This is simply a consequence of these methods making significantly fewer updates. In spite of this, the resultant effect of increasing the bundle size is a larger range of hyperparameters that produce good generalisation performance.

The results from the label noise experiments are shown in Figure 4.6. For $p = 0.1$ the results closely mirror those where no label noise is used, shown in Figure 4.5, however, here all models achieve roughly $0 - 5\%$ worse test performance. When the noise is increased to $p = 0.5$ the test error increases drastically by $0 - 25\%$. In both cases where $p = 0.1$ and $p = 0.5$, increasing N obtains better results for a larger range of values of η and r. Interestingly, using $N = 3, 5$ results in the best performance when $p = 0.5$. This is somewhat expected since using a larger bundle size means more samples are used to calculate each parameter update and hence helps reduce the effect of the label noise.

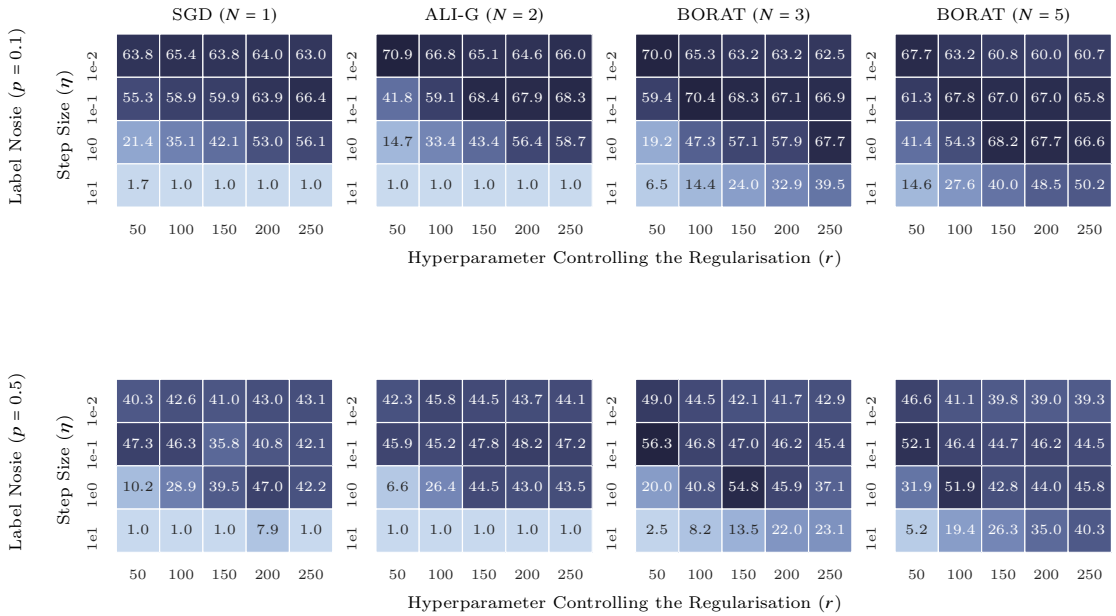


Figure 4.6: *Test accuracy of SGD, ALI-G, BORAT3, and BORAT5's robustness to hyperparameters when trained on CIFAR-100 with noisy labels. Here we give results with two different levels of noise* $p = 0.1$ *(upper row) and* $p = 0.5$ *(lower row). For readability purposes, the colour encodes test accuracy, where darker colours correspond to higher values. Increasing* N *allows for higher learning rates and greater levels of regularisation to be used. Additionally, when the level of noise is high* ($p = 0.5$*, lower row), BORAT significantly outperforms SGD and ALI-G.*

Figure 4.7 details the robustness of SGD, ALI-G and BORAT for the Tiny ImageNet experiments. We provide results for the Adam optimiser in Appendix B.2. These results show BORAT offers improved robustness on multiple data sets as we recover similar performance to CIFAR-100. For the CE loss increasing N allows for slightly higher learning rates and greater levels of regularisation to be used for the CE loss. For the SVM loss BORAT produces models with reasonable accuracy for a few of the hyperparameters combinations with $\eta \in \{0.1, 1.0\}$ as opposed to SGD and ALI-G which produce poor results for all learning rates excluding $\eta = 1.0$. Consequently, when training with the SVM loss, BORAT produces a drastically better model than SGD and ALI-G. This happens despite BORAT making significantly fewer updates of $\boldsymbol{w}_t$ in the 200 epochs.

Figure 4.7: *Comparison of SGD, ALI-G, and BORAT's robustness to hyperparameters on the Tiny ImageNet data set. Here we investigate the effect of increasing the bundle size when in use with the CE and Hinge losses. Colour represents validation performance, where darker colours correspond to higher values. Where the CE loss is used BORAT produces an increase in the range of hyperparameters that result in high accuracy models. For the SVM loss BORAT is the only optimiser to produce highly accurate results.*

Across all six experiments BORAT consistently produces high accuracy models for a larger range of hyperparameter combinations than SGD or ALI-G. This is particularly pronounced for the SVM loss experiments where the decreased sensitivity of BORAT makes all the difference between finding hyperparameters that result in a high accuracy model and not. Consequently, in comparison to its competitors, BORAT is systematically more robust to the choice of learning rate and regularisation hyperparameter, and also offers better generalisation more often than not. Therefore, we particularly recommend using BORAT for tasks where hyperparameter tuning is a highly time consuming task.

4.6 Discussion

In this chapter we have introduced BORAT a bundle method designed for the optimisation of DNNs capable of interpolation. BORAT offers similar performance to ALI-G while boasting increased robustness by way of a greater bundle size. BORAT produces high accuracy models for a larger range of hyperparameters and is particularly effective when using the multi-class hinge loss or in the presence of label noise. Our results suggest that BORAT is presently the most robust single hyperparameter optimisation method for deep neural networks.

It may be possible to modify BORAT so that it can make a parameter update after each gradient evaluation by cleverly selecting how linear approximations are added and removed from the bundle. If done correctly, this may lead to a further increase in the speed of convergence. However, we leave this to future work. When keeping the total number of gradient evaluations constant, a downside of using a larger bundle size for BORAT is that the number of parameter updates decreases. Despite this, in our experiments, BORAT is able to obtain good results within the same budget of passes through the data. Notably, this also means that BORAT has a time per epoch comparable to that of adaptive gradient methods. Another potential criticism of BORAT is that its memory footprint grows linearly with the bundle size. However, in practice, our results show that using a bundle size of three, which corresponds to the same memory cost as Adam, is often sufficient to obtain improved robustness. Finally, the applicability of BORAT can be limited by the required assumption of interpolation. While we argue that this interpolation property is satisfied in many interesting use cases, it may not hold true for one or more of the following confounding factors: (i) limited size of the neural network; (ii) large size of the training data set, which is becoming increasingly common in machine learning; and (iii) complexity of the loss function, such as in adversarial training. Furthermore, the concept of interpolation itself is ill-defined for unsupervised tasks. In the next chapter we generalise ALI-G to non-interpolating settings.

Chapter 5

Faking Interpolation Until You Make It (ALI-G+)

5.1 Introduction

In this chapter, we propose an optimisation method for the non-interpolating setting. We introduce a novel extension of ALI-G (Berrada et al., 2020) to tasks where the interpolation property does not hold. As we no longer have access to the optimal loss values *a priori*, we instead estimate it for each sample online. To realise this, we introduce a simple but highly effective heuristic for approximating the optimal value based on previous loss evaluations. We provide rigorous experimentation on a range of problems. From our empirical analysis we demonstrate the effectiveness of this approach, which outperforms other single hyperparameter optimisation methods.

Our algorithm is based on the observation that any non-interpolating problem can be made to satisfy the interpolation property once a point that minimises the training objective is known. One simply modifies each loss to be the pointwise maximum of the loss function and its value at the optimal point. Moreover, one only requires the knowledge of this optimal loss value for every example and not the location of the minimiser in parameter space. Hence, if one is able to approximate

the loss values at an optimal point with reasonable accuracy, one should be able to replicate the desirable characteristics of algorithms such as ALI-G and SPS (Loizou et al., 2021). Specifically, an algorithm with a single fixed hyperparameter that is easy to tune, and has strong generalisation performance. We present an optimisation method that approximates the optimal function values online using a heuristic in combination with a Polyak step size. We name our algorithm Adaptive ALI-G (ALI-G+), as it makes use of a modified ALI-G step size iteratively to update the parameters.

We conduct a thorough empirical evaluation of ALI-G+ on a variety of tasks against strong baselines. We provide results for matrix factorisation, binary classification using RBF kernels, image classification on the SVHN, CIFAR, Tiny ImageNet and ImageNet data sets, review classification and next character prediction. These tasks are designed to provide a mix of non-interpolating and interpolating problems. In all cases ALI-G+ outperforms all other single hyperparameter methods, often by a significant margin. These results demonstrate that estimating the optimal loss value online is an effective approach for selecting the step size.

5.2 Training in Non-Interpolating Settings

While the interpolation setting has received a lot of attention from recent work, many interesting problems do not satisfy this assumption. This could be for any of the following reasons: i) the model size could be limited due to hardware or power constraints, such as for embedded devices; ii) the data set is very large, for example, the vast majority of models trained on the ImageNet data set (Deng et al., 2009) do not achieve zero training loss; iii) complexity of the loss function, such as in adversarial training; iv) label noise can make interpolation impossible by creating a one to two mapping between inputs and labels. Thus, we think this setting is deserving of bespoke optimisation algorithms that are easy to use and

produce strong generalisation performance.

5.2.1 Motivation.

Our algorithm is motivated by trying to approximate $\ell_z(\boldsymbol{w}_\star)$ online, and as a result recover interpolation. Thus, we introduce a scalar $\tilde{\ell}_z^k$ to store our estimate for each example in the training set. We refer to these scalars as approximate optimal values (AOVs) and the superscript k indicates how many times the approximation has been updated. Our algorithm alternates between two "steps" i) using the current approximation of the optimal loss $\tilde{\ell}_z^k$ to inform the step size, (see Algorithm 3); and ii) improving the approximations based on the best previous iterates, (see Algorithm 4). We describe these "steps" in detail in the following two sections.

5.2.2 Related work.

The idea of trying to approximate loss values online in combination with a Polyak step size has also recently been explored by Davtyan et al. (2022) who use a global approximation to the loss based on Kalman filtering. Conversely Gower et al. (2021) present and number of different schemes using this idea. These include an algorithm they name Moving TArgetted Stochastic Polyak Step or MOTAPS, which like ALI-G+ constructs a estimate for each training example. MOTAPS enjoys some nice theoretical properties, such as provable convergence on convex problems. However, the authors only show results for deep learning problems when interpolation holds or approximately holds, so it unclear how well their method works in practice.

5.2.3 Updating the Parameters.

ALI-G+ uses the same stochastic version of the Polyak step size as ALI-G Equation (3.16). However, we replace the optimal loss value $\ell_z(\boldsymbol{w}_\star) = 0$ with its current approximation $\tilde{\ell}_z^k$. Hence, at time t the ALI-G+ algorithm uses the following weight

update:

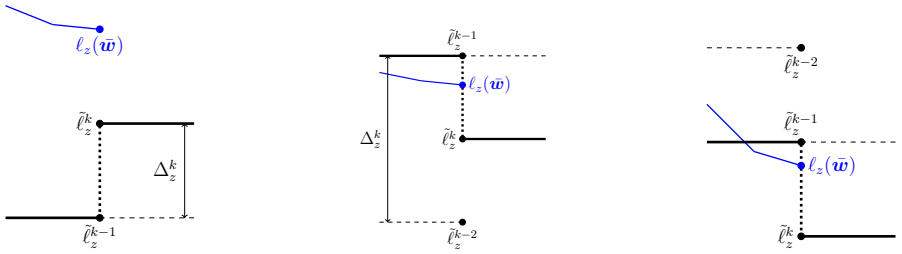
$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \gamma_t \boldsymbol{g}'_t, \tag{5.1}$$

$$\gamma_t \triangleq \max\left\{\min\left\{\eta, \frac{\ell_{z_t}(\boldsymbol{w}_t) - \tilde{\ell}_z^k}{\|\nabla \ell_{z_t}\|^2}\right\}, 0\right\}. \tag{5.2}$$

We define $\boldsymbol{g}'_t \triangleq (\nabla \ell_{z_t}(\boldsymbol{w}_t) + \lambda \boldsymbol{w}_t)$, where η and λ are the hyperparameters controlling the maximum step size and weight decay amount, respectively. The loss, AOV and gradient values are averaged over the mini-batch. As we do not require the interpolation assumption to hold, we do not need to use the constraint based regularisation of ALI-G, and can simply make use of weight decay, which allows for easy comparison with other algorithms. It is worth noting here that the max with 0 is no longer redundant as there is no guarantee that $(\ell_{z_t}(\boldsymbol{w}_t) - \tilde{\ell}_z^k)$ will be positive. Without this positivity constraint a negative step size could be used resulting in a gradient ascent step. Moreover if $\ell_{z_t}(\boldsymbol{w}_t)$ is already lower than its AOV $\tilde{\ell}_z^k$ then we have already completed our goal for this sample in this "step". In the next "step" we can then update this AOV to a lower value, which we describe in the next section. The full procedure for updating the parameters given the AOVs is outlined in Algorithm 3. In Appendix D.1 we detail some unsuccessful parameter update schemes.

Algorithm 3 ALI-G with AOVs

1: **Input**: time horizon T, initial point $\boldsymbol{w}_0$, maximum step size η, AOVs $\tilde{\boldsymbol{\ell}}$ and λ.
2: **for** $t = 0, ..., T-1$ **do**
3: Sample $z_t \in \mathcal{Z}$, $\ell_{z_t}(\boldsymbol{w}_t)$, $\nabla \ell_{z_t}(\boldsymbol{w}_t)$
4: Set $\gamma_t = \max\left\{\min\left\{\eta, \frac{\ell_{z_t}(\boldsymbol{w}_t) - \tilde{\ell}_z^k}{\|\nabla \ell_{z_t}\|^2}\right\}, 0\right\}$
5: $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \gamma_t(\nabla \ell_{z_t}(\boldsymbol{w}_t) + \lambda \boldsymbol{w}_t)$
6: Return $\bar{\boldsymbol{w}} \approx \text{argmin}_{t \in \{1,...,T\}} \{f(\boldsymbol{w}_t)\}$



(a) Increasing an AOV (b) Decreasing an AOV (c) Consecutive Decreases

Figure 5.1: *The possible AOV updates of a single sample. The thick black line represents AOV values. Blue line depicts $\ell_z(\bar{\boldsymbol{w}})$, Thin black line represents previous AOV value. Panel (a) shows the update if an AOV has not been reached. Panel (b) conversely shows the process for a AOV that has been reached for the first time, the AOV is lowered half way to the previous value. Panel (c) shows samples that have reach their AOV for consecutive sections have the same absolute decrease in value applied.*

5.2.4 Updating the AOVs.

To replicate the performance of algorithms for interpolation we want the approximation $\tilde{\ell}_z^k$ to tend towards $\ell_z(\boldsymbol{w}_\star)$ throughout training. Due to the stochastic and non-convex nature of training neural networks it is impossible to guarantee this behaviour. However, we present a simple scheme for updating the AOVs that demonstrates strong empirical performance as shown in Section 5.4. This scheme is inspired by both the curriculum learning literature and the work of Hazan and Kakade (2022), which presents a theory for a similar scheme for the convex and deterministic settings. The AOV update scheme is designed to be both reactive and optimistic. By reactive we mean that if a specific sample returns a constant loss its AOV will tend toward this value. By optimistic we mean that the AOVs are updated to a lower value than the current best loss value for this sample, in the hope that a further decrease in loss is possible. In practice the loss values of specific samples can fluctuate throughout training especially when data augmentation is used. However, the step size is calculated on a batch of data, and hence is relatively robust to this noise.

The AOV update scheme is as follows: we store the vectors $\tilde{\boldsymbol{\ell}}^k, \boldsymbol{\Delta}_z, \boldsymbol{\ell}(\bar{\boldsymbol{w}})$ containing $(\tilde{\ell}_z^k, \Delta_z, \ell_z(\bar{\boldsymbol{w}}))\forall z \in \mathcal{Z}$, where $\bar{\boldsymbol{w}} = \text{argmin}_{t\in\{0,\dots,T\}} \{f(\boldsymbol{w}_t)\}$ in memory and Δ_z is

the last positive update to the corresponding AOV. The AOVs $\tilde{\ell}_z^k$, are initialised to our known lower bound on the loss B. $\boldsymbol{\Delta}_z$ is initialised to zero. The training duration is split into K equal sections each with length T. During each of these sections we keep the AOVs fixed and try to get a good estimate of $\ell_z(\bar{\boldsymbol{w}})$ for each example. After each of the K sections we update all AOVs simultaneously. Each AOV is updated depending on whether it has been "reached", that is, if $(\ell_z(\bar{\boldsymbol{w}}) \leq \tilde{\ell}_z^k)$ is true. In both cases we are optimistic that the loss can be decreased further from its current value. Hence, if an AOV hasn't been reached it is updated by simply averaging $\ell_z(\bar{\boldsymbol{w}})$ and $\tilde{\ell}_z^k$, see Figure 5.1 (a). This increases this AOV to halfway between the loss at the best point visited and its current value. However, if $(\ell_z(\bar{\boldsymbol{w}}) \leq \tilde{\ell}_z^k)$ we instead try decreasing $\tilde{\ell}_z^k$ halfway to the last value that was reached $\tilde{\ell}_z^{k-1}$, Figure 5.1 (b). If the z^{th} AOV is reached again in successive sections we reduce $\tilde{\ell}_z^k$ each time by the same magnitude, see Figure 5.1 (c). Thus, even if an AOV is incorrectly updated to a value higher than $\ell_z(\boldsymbol{w}_\star)$ it can easily be corrected by consecutive reductions. Lastly we ensure AOVs are never decreased below the lower bound B. Hence, for non-negative losses AOVs are always positive.

This scheme can be thought of as defining a curriculum for the non-interpolating setting. Rather than first focusing on easy examples, at the beginning all examples are treated equally. After time, examples that are identified as hard are given less importance, and the optimiser does not try to optimise these examples further. It makes sense to not focus on the hardest examples as we know we cannot achieve zero loss on all samples simultaneously. In Appendix D.1 we detail some unsuccessful AOV update schemes.

Algorithm 4 ALI-G+ Algorithm

1: **Input**: time horizon T_{max}, $K = 5$, $\bar{\boldsymbol{w}}_0$ and $\Delta_z^0 = 0, \tilde{\ell}_z^0 = B, \forall z \in \mathcal{Z}$ and λ.
2: **for** epoch $k = 1, ..., K$ **do**
3: Run Algorithm 3 with $\bar{\boldsymbol{w}}^{k-1}$, $\frac{T_{max}}{K}$, $\tilde{\boldsymbol{\ell}}^k$, η and λ to obtain $\bar{\boldsymbol{w}}^k$.
4: **for** $z \in \mathcal{Z}$ **do**
5: **if** $\ell_z(\bar{\boldsymbol{w}}) \leq \tilde{\ell}_z^k$ **then**
6: $\tilde{\ell}_z^{k+1} \leftarrow \max\{\frac{\tilde{\ell}_z^k + \tilde{\ell}_z^{k-1}}{2}, \mathrm{B}\}$
7: $\Delta_z^{k+1} \leftarrow \Delta_z^k$
8: **else**
9: $\tilde{\ell}_z^{k+1} \leftarrow \frac{\tilde{\ell}_z^k + \ell_z(\bar{\boldsymbol{w}})}{2}$
10: $\Delta_z^{k+1} \leftarrow \frac{\tilde{\ell}_z^k - \ell_z(\bar{\boldsymbol{w}})}{2}$
11: $k \leftarrow k + 1$
12: Return $\bar{\boldsymbol{w}}_K$

5.2.5 Implementation Details.

For the above scheme to work well, it is important that the AOVs are not updated too frequently, as this can lead to them trending towards $\ell_z(\bar{\boldsymbol{w}})$ too fast. However, it is also important that the AOVs are updated a sufficient number of times so they can approximate $\ell_z(\boldsymbol{w}_\star)$, if $\ell_z(\boldsymbol{w}_\star)$ is large. We find $K = 5$ provides a good balance between these considerations and fix K to this value. However, ALI-G+ is relatively robust to the choice of K and produces good results for $k \in [5, 10]$. In the Appendix D.3 we provide additional results for $K \in \{3, 10, 20\}$. Furthermore, to save computation we i) avoid calculating $f(\boldsymbol{w}_t)$ exactly and instead approximate this online during each epoch and ii) we use $\boldsymbol{w}_T^{k-1}$ in the place of $\bar{\boldsymbol{w}}_{k-1}$ in line 3 of Algorithm 4. This results in ALI-G+ having a similar run time to SGD, where the only extra computation is the updating of the vectors $\tilde{\boldsymbol{\ell}}^k, \tilde{\boldsymbol{\ell}}^{k-1}, \boldsymbol{\ell}(\bar{\boldsymbol{w}})$ and evaluating the norm of the gradients. As is common practice we report the results of the model with the best validation performance found during training.

5.2.6 Data Augmentation.

Data augmentation can be thought of in two ways. First, it increases the size of the data set by adding new examples that are simply transformed versions of others. Second, it makes online alterations to the original number of examples. As ALI-G+ is designed for the optimisation of non-interpolating problems, which often have large data sets, we choose to view data augmentation in the second way and save only a single AOV for all possible augmentations. When viewing data augmentation in the first way, training regimes where the number of epochs is less than the number of possible transformations would only visit each example less than once on average. Hence, approximating the optimal value would be challenging. Moreover, for many common data augmentation transforms, such as random crops of images, we would expect the optimal loss value to be highly correlated between the same example under different versions of the transformation. To support this claim we calculate the loss value of all possible crops for a subset of 5000 images chosen from a selection of common data sets. We find empirically, at the start of training that the variance between loss values is on average 20 times lower for the different crops of the same image compared to randomly chosen images. During training we observe this ratio drops to 5 times lower.

5.3 Worst Case Behaviour

We provide the following bound on ALI-G+ for smooth and convex functions. As ALI-G+ is computationally identical to ALI-G before the first AOV update we can modify the result of Berrada et al. (2020) to give a worst case bound.

Theorem 3 (Worst Case Bound - Smooth and Convex)**.** *We assume that for every* $z \in \mathcal{Z}$, ℓ_z *is convex and* β*-smooth. Let* $\boldsymbol{w}_\star$ *be a solution of* $(\mathcal{P})$ *such that* $\forall z \in \mathcal{Z}$, $\ell_z(\boldsymbol{w}_\star) \leq \varepsilon$*. Further assume that* $\eta \leq \frac{1}{2\beta}$ *and* $\lambda = 0$*. Then if we apply ALI-G+*

with a maximal learning rate of η to f, we have:

$$\min_t f(\boldsymbol{w}_t) - f_\star \leq \frac{\|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2}{\eta(\frac{T}{K}+1)} + \epsilon. \tag{5.3}$$

See Appendix C.1 for derivation. We provide a similar bound for the smooth and strongly convex setting in Appendix C.2.

5.4 Experiments

In this section we test the hypothesis that a Polyak-like step size in combination with AOVs can produce high accuracy models in the non-interpolating setting. We investigate this through rigorous experiments comparing ALI-G+ against a wide range of single hyperparameter optimisation algorithms on a variety of problems.[1] The problems include both non-interpolating and interpolating settings. We start with relatively simple problems such as matrix factorisation and binary classification using RBF kernels. We then consider the training of deep neural networks on popular image classification benchmarks. Here we use small models for two reasons: i) to induce non-interpolation and ii) to be able to investigate a wider range of data sets, hyperparameters and baselines. Ideally we would have included results for large models trained on even larger data sets, to ensure non-interpolation. However, this was not within the limits of our computational resources. We have tried to perform as thorough a set of experiments as possible and our experimental setup is on par with, in fact often exceeds, prior work in this area of machine learning in terms of data sets and baselines considered (Vaswani et al., 2019; Loizou et al., 2021).

We show that our approach scales to large problems by providing results on the ImageNet data set. Furthermore, we do not just show results for computer vision data sets but also for two NLP tasks, highlighting the flexibility of our approach. All experiments are conducted in PyTorch (Paszke et al., 2017) and are performed

[1] code available at https://github.com/Alasdair-P/alig_plus

on a single GPU except for the ImageNet experiments which use two.

5.4.1 Simple Optimisation Benchmarks

Setting. We first demonstrate the performance of ALI-G+ on matrix factorisation and RBF Binary Classification using tasks detailed in Vaswani et al. (2019). The matrix factorisation task can be expressed as:

$$\min_{\boldsymbol{W}_1, \boldsymbol{W}_2} \mathbb{E}_{\mathbf{x} \in \mathbb{X}}[||\boldsymbol{W}_1 \boldsymbol{W}_2 \mathbf{x} - \boldsymbol{A}\mathbf{x}||^2], \tag{5.4}$$

where $\mathbb{X}$ is a data set of 1000 examples drawn from $\mathcal{N}(\mathbf{x}; 0, \boldsymbol{I})$, $\boldsymbol{A} \in \mathbb{R}^{10 \times 6}$ is randomly generated to have condition number 10^{10}, $\boldsymbol{W}_1 \in \mathbb{R}^{10 \times k}$, $\boldsymbol{W}_1 \in \mathbb{R}^{k \times 6}$, $\boldsymbol{A}$. The rank of the factorisation k is selected to be one of four different values resulting in two problems where interpolation holds and two where it does not. For the binary classification tasks with radial basis functions we use the mushrooms and ijcnn data set from the LIBSVM library of SVM problems (Chang and Lin, 2011). The mushrooms data set satisfies the interpolation assumption, whereas ijcnn does not.

Method. We compare ALI-G+ against Parabolic Approximation Line Search (PAL) (Mutschler and Zell, 2020) and a selection of the optimisation methods used in Vaswani et al. (2019). We additionally reuse their code for the baselines. These optimisation algorithms contain a collection of strong line search and adaptive gradient methods, all of which do not require a learning rate schedule. Additionally, the majority have a single step size hyperparameter which makes for fair comparison with ALI-G+ .

Results. The results of these experiments are shown in Figures 5.2 and 5.3. On the non-interpolating tasks (rank 1 and rank 4 matrix factorisation and binary classification on the ijcnn data set), ALI-G+ performs comparably to the best algorithms, specifically, PAL (Mutschler and Zell, 2020) and SLS (Vaswani et al., 2019). On the

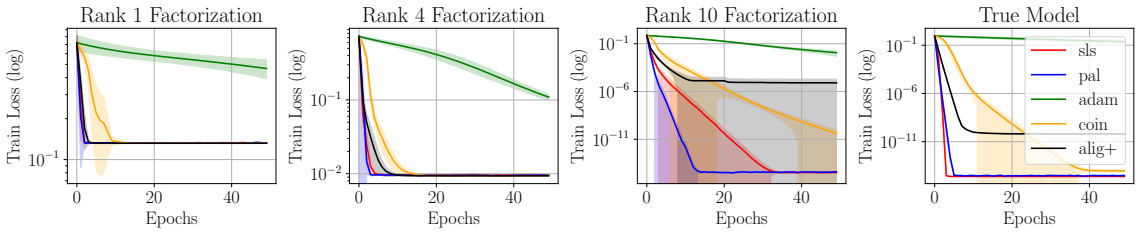


Figure 5.2: *Training performance on the matrix factorisation problem of Vaswani et al. (2019). In the settings where interpolation does not hold, namely the Rank 1 and Rank 4 problems, ALI-G+ quickly achieves the loss floor. For the Rank 10 and True Model problems ALI-G+ does not minimise the loss to machine precision, such as SLS (Vaswani et al., 2019) and PAL (Mutschler and Zell, 2020), however, it still provides rapid optimisation to at worst* $> 10^{-4}$.

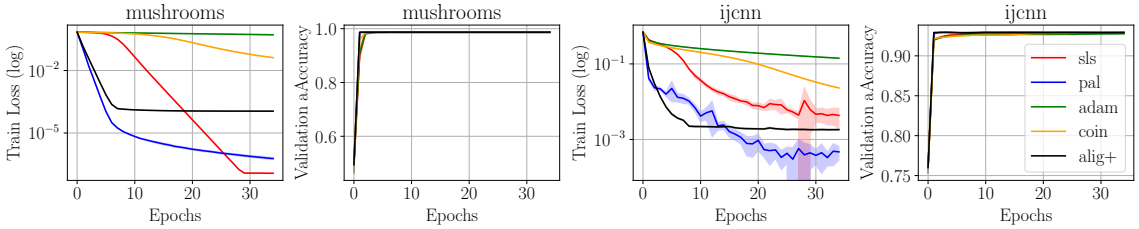


Figure 5.3: *Training and validation performance on the mushrooms and ijcnn data sets (Chang and Lin, 2011). On the mushroom data set, where interpolation holds, ALI-G+ fails to achieve the same training loss as the line search methods. However, in both non-interpolating and interpolating settings ALI-G+ obtains equally good validation performance as the best baseline.*

interpolating tasks ALI-G+ fails to minimise the training loss to machine precision like PAL and SLS. However, it attains the same validation performance, which is impressive as ALI-G+ is not designed for the interpolation setting. For these tasks using ALI-G would be a better option.

5.4.2 Image Classification Experiments

Setting. We run experiments on a broad range of image classification benchmarks. Specifically we use the SVHN (Netzer et al., 2011), CIFAR-10, CIFAR-100 (Krizhevsky, 2009) and Tiny ImageNet data sets. The SVHN and CIFAR data sets are comprised of 32x32 pixel RGB images. For the SVHN data set we use the split proposed in Berrada et al. (2020) resulting in 598k training, 6k validation and

26k test samples. SVHN and CIFAR-100 both have 10 classes and CIFAR-100 has 100. The Tiny ImageNet data set is more challenging and contains 100k training examples of 64x64 pixels split over 200 classes. For the Tiny ImageNet data set the ground truth labels of the test set are not freely available so we report validation scores instead. All images are normalised per channel and when data augmentation is used we apply standard random flips and crops. For the majority of data sets we present results with and without data augmentation. The exceptions being SVHN, which is not designed for data augmentation. For all data sets we make use of the cross entropy loss to train a small 8 layer ResNet (He et al., 2016) containing 90k parameters with 16 channels in the first layer. These tasks were chosen to give examples of i) interpolation (SVHN); ii) near interpolation (CIFAR-10) and iii) non-interpolation resulting from limited model size (CIFAR-100 and Tiny ImageNet).

Method. We compare ALI-G+ against PAL (Mutschler and Zell, 2020), ALI-G (Berrada et al., 2020), AdamP (Heo et al., 2021), SPS (Loizou et al., 2021), the optimisation methods used in Vaswani et al. (2019), SGD with a constant learning rate (SGD_{Const}) and SGD with a step learning rate schedule (SGD_{Step}). SGD_{Step} benefits from a manually tuned learning rate schedule developed by He et al. (2016) while all other methods have at most a single step size or maximum step size hyperparameter that is cross-validated as powers of ten. Hence, SGD_{Step} requires far more tuning and does not provide a fair comparison. However, it is included for completeness. For all optimisation algorithms the problem regularisation hyperparameter is selected from $\lambda \in \{10^{-3}, 10^{-4}, 10^{-5}, 0\}$. ALI-G uses constraint based regularisation, (see section 3.9.1); r was selected from $r \in \{50, 100, 200, \infty\}$. All other hyperparameters are left at their default values. SGD_{step} we use the learning rate schedules detailed in He et al. (2016). See Table D.2 in Appendix D.4 for more details of hyperparameters used. We reuse the schedule proposed for the CIFAR data sets for SVHN and Tiny ImageNet, reducing the learning rate by a factor of 10

both halfway and three quarters through training. A fixed batch size of 128 and an epoch budget of 200 are used for all experiments. As is common for deep learning experiments we accelerate SGD, ALI-G and ALI-G+ with a Nesterov momentum of 0.9. SLS_{Polyak}, Adam and Adabound also include momentum-like terms which we leave at their default settings. We performed 3 runs for all experiments and report the average.

	SVHN	CIFAR-10		CIFAR-100		Tiny ImageNet	ImageNet
	Test Acc (%)					Val Acc (%)	
Model	Small ResNet						ResNet18
Data Aug	No	No	Yes	No	Yes	No	Yes	Yes
SGD_{Step}	95.4	84.1	87.6	51.0	59.6	39.8	43.2	71.1
SGD_{Const}	94.2	80.6	87.0	49.0	57.3	36.6	41.3	57.5
$ALI-G$	93.8	80.6	86.2	47.5	57.9	35.6	41.8	63.7
SPS	93.9	81.1	86.8	43.7	53.1	20.5	23.8	63.9
$Adabound$	93.1	75.6	85.2	44.0	55.4	34.2	40.1	62.9
$Adam$	94.0	79.7	86.0	48.1	56.2	35.9	41.3	62.6
$AdamP$	93.8	79.9	85.9	47.6	58.2	36.3	41.7	63.5
$Coin$	92.1	75.5	84.1	42.4	54.0	31.0	36.2	61.5
SLS_{Armijo}	93.0	81.5	85.7	31.6	42.0	11.2	11.1	63.2
$SLS_{Goldstein}$	92.3	78.3	86.4	45.5	57.2	33.0	40.4	62.6
SLS_{Polyak}	93.5	79.8	85.9	43.6	54.0	31.3	38.3	62.7
PAL	94.2	81.5	86.7	39.8	57.0	35.3	40.8	63.6
ALI-G+	**95.5**	**85.0**	**87.2**	**56.1**	**59.4**	**39.8**	**42.6**	**67.8**

Table 5.1: *Accuracies of optimisation methods on a selection of standard image classification data sets. The model and data set combinations have been chosen to include both interpolating and non-interpolating tasks. The standard deviation of the accuracy was at most* 0.3 *for ALI-G+ . SGD_{step} is the only method to benefit from a manually designed step size schedule. All other methods have at most one fixed step size hyperparameter. On these tasks ALI-G+ outperforms all other single hyperparameter methods, often by a large margin.*

Results. The accuracy of the best performing model for each optimisation method is shown in Table 5.1. On the tasks considered, ALI-G+ does at least as well as all other line search and adaptive gradient methods. On many tasks it outperforms all other methods by a significant margin. The main exception being on CIFAR-10 with

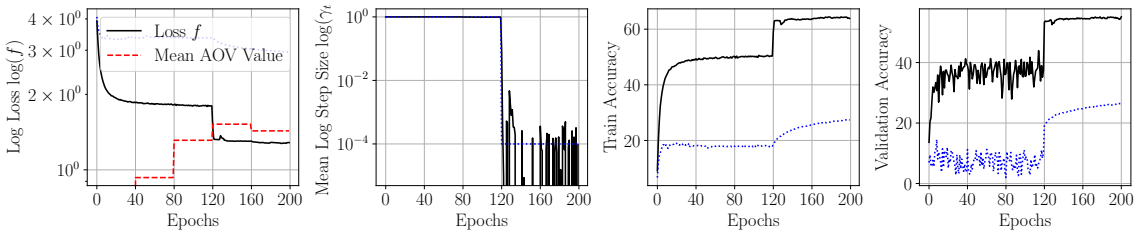


Figure 5.4: *The black solid lines show curves produced by training a small ResNet on CIFAR-100 using the ALI-G+ optimiser. Here* $\eta = 1.0$, $\lambda = 0.001$ *and no data augmentation was used. The AOVs are updated every 40 epochs. Until epoch 120 the mean loss is significantly higher than the mean AOV and thus the maximum step size* η *is used for the majority of updates. The blue dotted curve shows the results when the mean step size is used for all batches. The large difference in performance between these methods demonstrates the superiority of using a step size tailored to each batch. Appendix D.2 we provide additional training curves for ALI-G+ in a variety of settings.*

data augmentation where SGD_{Const} produced similar test accuracy. The dominant performance of ALI-G+ shows the lack of strong algorithms for non-interpolating settings. The performance benefit of ALI-G+ is most notable when the interpolation property is far from satisfied or when data augmentation is not used. For example, on the challenging Tiny ImageNet data set ALI-G+ produces validation accuracy 4% higher than the next best. Empirically we observe that ALI-G+ is almost twice as fast as the line search methods, significantly faster than Adam, and has a run time within a couple of percent of SGD. Typical training curves for ALI-G+ are shown in Figure 5.4. We present results for ALI-G+ with different values of K and a global step size in Appendix D.3

5.4.3 Large Image Classification Experiments

Setting. The ImageNet data set (Deng et al., 2009) contains 1.2M large RGB images of various sizes split over 1000 classes. For our experiments we use the following data augmentation. All images are normalised per channel, randomly cropped to 224x224 pixels and horizontal flips are applied with probability 0.5. For validation a centre crop is used and no flips are performed. For ImageNet the ground

truth labels are not freely available so we report validation scores instead. We train a ResNet18 containing 11.7M parameters (He et al., 2016). Due to the large number of images and data augmentation the interpolation assumption does not hold. We opted to not use a larger model to save on computation requirements and to ensure interpolation was not approximately satisfied.

Method. Due to computational constraints for all methods we reuse the best hyperparameters identified on Tiny ImageNet for the ImageNet experiments. However, the batch size is increased to 256 and the epoch budget is reduced to 90. For SGD_{step} we use the learning rate schedule described in He et al. (2016). Note, different learning rate schedules are suggested for CIFAR and ImageNet, again highlighting the weakness of using SGD_{step} where a good learning rate schedule is not known in advance.

Results. The validation accuracy of each optimisation method is shown in the last column of Table 5.1. On this task, ALI-G+ outperforms all line search and adaptive gradient methods by at least 3.5%. Additionally, ALI-G+ was significantly quicker to train than two of the next best performing methods. SLS_{Armijo} and PAL took 20 and 12 hours longer to train than ALI-G+ , respectively. These results show the advantage of ALI-G+ for training on large data sets over comparable techniques, especially line search methods.

Runtime and Memory Analyses. In Table 5.2 we include the run time for training a ResNet18 split across two NVIDIA TITAN XP GPUs on ImageNet. ALI-G+'s run time is comparable to adaptive gradient methods and faster than line search approaches. During training ALI-G+ requires that an additional $3|\mathcal{Z}|$ floats are stored where $|\mathcal{Z}|$ is the size of the training set. This extra memory requirement may seem large, however, in many scenarios $3|\mathcal{Z}|$ is orders of magnitude times smaller than the number of parameters of the model d. For a concrete example, when

training the ResNet18 on ImageNet, $3|\mathcal{Z}| \approx 3.8M$, however, ResNet18 requires 11.7M floats for the parameters alone, at least 11.7M for gradients and then roughly the same again for storing activations and their gradients. Thus, the additional memory requirement of ALI-G+ is negligible.

Optimiser	SGD	ALI-G	SPS	Adabound	Adam	AdamP	Coin	SLS	PAL	ALI-G+
Time (h)	32	34	33	33	35	35	34	53	47	35

Table 5.2: *Wall clock time for training ResNet18 on ImageNet with various single hyperparameter methods. Here ALI-G+ is as fast as adaptive gradient methods.*

5.4.4 NLP Experiments

Setting. For NLP Experiments we consider two tasks. The first is binary classification of reviews on the IMDB data set using a bi-directional LSTM. The second is the training of a Recurrent Neural Network (RNN) for character-level language modelling on the Tolstoi War and Peace data set, which forms part of the DeepOBS benchmark (Schneider et al., 2019). The bi-directional LSTM has 1 layer and the RNN has 2 layers. Both models have 128 hidden units per layer. By selecting these models the interpolation property is satisfied on the IMDB data set but not on the Tolstoi data set.

Method. We compare ALI-G+ against the majority of the algorithms used in section 5.4.2. However, we use a slightly modified cross-validation scheme; each optimiser's step size or maximum step size hyperparameter is again cross-validated as powers of ten. The weight decay amount λ was selected from $\{0.01, 0.001\}$ for the IMDB classification task, and was not applied to biases. For this task a batch size of 128 and an epoch budget of 100 was used. In contrast, no regularisation, a batch size of 50 and an epoch budget of 150 was used for Tolstoi character prediction.

Results. On the easy IMDB review classification task a large number of the optimisation methods achieved close to zero training loss and similar accuracies. The best performing of these were Adam and ALI-G+ , which attained a test accuracy of 87.9% and 88.0%, respectively. For the harder character prediction task using the Tolstoi data set ALI-G+ was the best performing algorithm by over 1%. The full results are shown in Table 5.3. These results reinforce i) that ALI-G+ consistently achieves highly competitive results in a wide range of settings; and ii) in the non-interpolating setting ALI-G+ is particularly effective.

Data Set	Model	SGD_{const}	Adabound	Adam	Coin	SLS_{Armijo}	$SLS_{Goldstein}$	SLS_{Polyak}	PAL	ALI-G+ (K=5)	ALI-G+ (K=10)
IMDB	LSTM	87.5	82.7	87.9	87.6	73.4	78.6	67.1	85.1	**88.0**	87.7
Tolstoi	RNN	49.4	41.7	57.9	56.9	30.0	39.0	31.3	52.9	**59.7**	59.4

Table 5.3: *Test accuracy of single hyperparameter optimisation methods on NLP data sets. For both data sets ALI-G+ is the best performing task with Adam a close second. However, on the relatively easy IMDB review classification task a large number of the optimisation methods achieved close to zero training loss and similar test accuracy.*

5.5 Discussion

In this chapter we have introduced ALI-G+, an optimisation algorithm designed for settings where interpolation does not hold. In run time and generalisation performance ALI-G+ demonstrates its effectiveness outperforming a wide range of modern neural network optimisation techniques on standard image classification benchmarks. We now briefly discuss three directions to explore further with ALI-G+. The first would be to characterise the conditions where ALI-G+ offers provable convergence to the global optimum. In the stochastic, Lipschitz continuous and convex

setting this would require additional assumptions on the data set and loss. However, ALI-G+ is designed for training deep neural networks where convergence proofs are not practical. The second idea is a modification to ALI-G+ when data augmentation is used. Here one could make use of a convenient distribution, to model the spread of $\ell_z(\bar{\boldsymbol{w}})$ given by the data augmentation. When updating the AOVs one could then use a lower confidence bound on this distribution. Finally, a promising direction that could be used to extend ALI-G+ would be its application to distillation, where the teacher network could be used to generate AOVs for the student. While these ideas seem like natural extensions we instead next turn our attention to the training of binary neural networks, which is the focus of the next two chapters.

Chapter 6

Binary Neural Networks

6.1 Motivation

As mentioned in Chapter 2, there is a trend towards the use of increasingly larger neural networks. Leveraging exceptionally large neural networks has led to some of the most exciting breakthroughs in machine learning. For example, GPT-3 (Brown et al., 2020), a very large transformer architecture, redefined the state of the art for language modelling. However, large models with billions of parameters have inherent limitations. Due to their computational costs, training large neural networks requires large, expensive, and power hungry hardware. For example, the largest version of GPT-3 required a staggering 10^{23} floating-point operations during training (Brown et al., 2020). Additionally, specific hardware can be required at inference time to run large models at reasonable speeds. This dependence on specialised hardware presents two major drawbacks for using large models. First, it restricts their use in power, size, or compute limited settings such as mobile devices and remote sensing applications. For some use cases it may be possible to offload the heavy computation to the cloud. However, this depends on the bandwidth and stability of the connection and criticality of the application, and thus is not always possible. Second, even in applications where dedicated hardware is not a limiting factor, the

energy usage of very large overparameterized models can still be significant, and this conflicts with a push to a more sustainable future. These limitations have led to significant work to reduce the memory and computational costs of neural networks while retaining strong generalisation performance. Many different methods have been proposed to this end, including distillation, efficient architectural choices, network pruning, and various compression and quantisation techniques. At the moment these are all active areas of research and it remains unclear which provide the best results (Neill, 2020).

6.2 Quantised Neural Networks

Quantised neural networks constrain nearly all their parameters to be an element of a finite set $\mathcal{Q}$, where the number of elements of $\mathcal{Q}$ is small, typically a power of 2. This reduces the memory required to store the network and for some choices of $\mathcal{Q}$ can result in more efficient computations within the parameterised layers of the network. This makes these networks well-suited to deployment on devices where memory or computation is limited.

6.3 Binary Neural Networks

Binary neural networks (BNN) are an extreme form of network quantisation where $\mathcal{Q} = \{-1, 1\}$. Binary parameters require only a single bit to store, compared to standard 32 bit floats. Thus BNN can offer up to a 32-fold reduction in memory requirements, depending on what fraction of the parameters are quantised. Binarized Neural Networks or Fully binary neural networks (FBNN) take this idea further, restricting all inputs to the parameterised layers of the network to the set $\{-1, 1\}$ as well. The majority of the computational cost of neural networks is taken by computing matrix multiplication within the parameterised layers. As all numbers within FBNN parameterised layers are binary they allow the majority of floating

point arithmetic operations to be replaced by faster bit-wise operations, which can be performed on cheaper hardware. In these models, a dot product can be implemented using a bit-wise XNOR operation followed by a bit count operation (Hubara et al., 2016). This is in contrast to conventional floating point models where a dot product requires numerous floating point multiplications and accumulation operations. On devices with less parallelism such as many CPUs, converting a network to an FBNN can offer a drastic 50 fold increase in inference speed (Rastegari et al., 2016).

6.4 Training Binary Neural Networks

In this section we detail a common set of assumptions when considering BNNs and we state how this paradigm impacts research on their training. Typically a BNN is desired to efficiently perform some inference task on some lightweight hardware. For example it might be advantageous to use an FBNN to control the steering and navigation in a self driving electric car due to the lower energy usage. It is assumed that the model performing this task does not need to be trained in-place on the lightweight hardware. Instead, a setting where extra computational resources are available can be used. Revisiting our example, a high performance computing cluster could be used by the company developing the navigation system. Once trained, the final BNN is downloaded onto the many lightweight devices executing the task, the car in our example. Hence it is normally assumed when training BNNs extra resources can be used and thus most works use floating point parameters during this phase. Additionally, as inference or test performance is the focus the training cost and time are not the number one concern when considering BNNs. While this is the dominant paradigm in the literature, there are of course many interesting exceptions.

For some less extreme choices of $\mathcal{Q}$, such as using 8 bit floats to encode each

parameter, it is possible to achieve good results by first training a conventional real valued network and then converting the chosen parameters to be elements of $\mathcal{Q}$. This has the advantage that it can be done in a data free manner, which may be useful if the training data are confidential. For a thorough review of this area we refer the interested reader to the work by Wang et al. (2018). For more extreme forms of quantisation, such as BNN, novel training schemes have been demonstrated to produce superior results Yuan et al. (2021). Due to the discrete nature of binary parameters, existing continuous optimisation techniques, such as those described in Chapter 2, are unsuitable for training BNNs. In this section we give an overview of the three main strategies for training these models.

6.5 Quantisation Scheme

The exact constraints on the parameters and activations within a quantised network is known as a quantisation scheme. Many quantisation schemes have been proposed in the literature, offering different benefits. We refer the interested reader to Gholami et al. (2021) for a review of this area. While the selection of a scheme is an active area of research, the focus of this thesis is the optimisation strategy of neural networks rather than their architectural design. Hence, we now introduce a canonical binary quantisation scheme which we use as a testbed for investigating the optimisation of FBNN.

6.5.1 Binary Activations

In order to ensure that the activations of an FBNN are binary, the sign function is applied to the inputs of parameterised layers, such as linear or convolutional layers:

$$\text{sign}(a) = \begin{cases} 1, & \text{for } a \geq 0, \\ -1, & \text{for } a < 0. \end{cases} \tag{6.1}$$

The gradient of the sign function is zero almost everywhere and hence in order to backpropagate gradients through the model it is necessary to approximate the sign function's derivative with a continuous alternative. A common choice for this approximation is the Straight Through Estimator, which approximates the derivative of $\text{sign}(a)$ in the backwards pass with:

$$\frac{\partial \text{sign}(a)}{\partial a} = \begin{array}{ll} 1, & \text{for } |a| \leq 1 \\ 0, & \text{for } |a| > 1 \end{array} . \tag{6.2}$$

Alternatively, Liu et al. (2020, 2018a) use the following approximation,

$$\frac{\partial \text{sign}(a)}{\partial a} = \begin{array}{lll} 2a+2, & \text{for} & -1 \leq a \leq 0 \\ 2-2a, & \text{for} & 0 \leq a \leq 1 \\ 0, & \text{for} & 1 \leq |a| \end{array} . \tag{6.3}$$

6.5.2 Binary Parameters

When considering BNN we use $\boldsymbol{w}^b$ to represent the subset of model parameters $\boldsymbol{w}$ that we want to take binary values. For clarity note that $\boldsymbol{w}^b$ does not always encode the final quantised values themselves. In other words, during an intermediate stage of training, $\boldsymbol{w}^b$ may refer to a real valued vector. However, this should be clear from the context. We aim to obtain a model with $\boldsymbol{w}^b \in \{-1, 1\}^p$. As is common practice we let some parameters retain real values. We denote the vector containing these unconstrained parameters with $\boldsymbol{w}^r \in \mathbb{R}^{d-p}$. The subset $\boldsymbol{w}^r$ typically includes the weights in the first and last layers of the network, all biases, and parameters of batch norm layers. Within this notation $\boldsymbol{w}$ is simply the concatenation of $\boldsymbol{w}^b$ and $\boldsymbol{w}^r$. Finally, we define the feasible set $\Phi \triangleq \{-1, 1\}^p \cup \mathbb{R}^{d-p}$ which represents all constraints on $\boldsymbol{w}$. Hence, to convert $(\mathcal{P})$ to express the problem of training a BNN or FBNN we set $\Omega = \Phi$. Thus, the task of training an FBNN can be formulated as

follows:

$$\boldsymbol{w}_\star \in \operatorname*{argmin}_{\boldsymbol{w}\in\Phi} f(\boldsymbol{w}). \tag{$\mathcal{B}$}$$

This is a highly non-linear non-convex mixed integer program and thus is NP hard in general, and hence does not permit an efficient exact solution.

6.5.3 Naive PSGD

Naively applying PSGD as described in equation (6.4) to the binary parameters would require an extremely large step size η_t and hence would not lead to stable optimisation. Thus, we next discuss how notable previous works have instead tackled optimising ($\mathcal{B}$).

$$\boldsymbol{w}^b_{t+1} = \operatorname{sign}(\boldsymbol{w}^b_t - \eta_t \nabla \ell_{z_t}(\boldsymbol{w}^b_t)). \tag{6.4}$$

6.5.4 The Straight Through Estimator Method (STE)

What has now become known as The Straight Through Estimator Method (STE) was first introduced in Courbariaux et al. (2015) as a method to train neural networks with binary weights and real valued activations. A year later Hubara et al. (2016) showed that this method could be used to train neural networks with both binary weights and activations. The STE method relies on introducing a second set of auxiliary parameters $\tilde{\boldsymbol{w}}^b \in \mathbb{R}^p$ used to represent the binary parameters during training. These auxiliary parameters are quantised via the sign function to calculate $\boldsymbol{w}^b$ before every forward pass.

$$\boldsymbol{w}^b_t = \operatorname{sign}(\tilde{\boldsymbol{w}}^b_t). \tag{6.5}$$

The gradient is then evaluated on the model with the binary parameters $\boldsymbol{w}^b_t$. However, this gradient is used to update the auxiliary parameters in the backwards pass.

This optimisation scheme at time step t can be succinctly described as follows:

$$\tilde{\boldsymbol{w}}^b_{t+1} = \Pi_{[-1,1]}(\tilde{\boldsymbol{w}}^b_t - \eta_t \nabla \ell_{z_t}(\boldsymbol{w}^b_t)), \tag{6.6}$$

$$\boldsymbol{w}^r_{t+1} = \boldsymbol{w}^r_t - \eta_t \nabla \ell_{z_t}(\boldsymbol{w}^r_t), \tag{6.7}$$

where $\Pi_{[-1,1]}$ is the projection on the the set $[-1,1]^p$. In other words $\tilde{\boldsymbol{w}}^b$ is used to accumulate gradients and allows successive gradients to change the sign of a parameter. In practice, the Adam update (3.12) is used in equation (6.6) and (6.7). Hence, $\boldsymbol{w}_b$ can be calculated using the following equations:

$$\tilde{\boldsymbol{w}}^b_t = \text{sign}(\boldsymbol{w}^b_{t-1}). \tag{6.8}$$

$$\hat{\boldsymbol{m}}_t = \frac{\boldsymbol{m}_t}{1-\beta_1^t}, \tag{6.9}$$

$$\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\nabla \ell_{z_t}(\tilde{\boldsymbol{w}}^b_t), \tag{6.10}$$

$$\hat{\boldsymbol{v}}_t = \frac{\boldsymbol{v}_t}{1-\beta_2^t}, \tag{6.11}$$

$$\boldsymbol{v}_t = \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\nabla \ell_{z_t}(\tilde{\boldsymbol{w}}^b_t)^2, \tag{6.12}$$

$$\boldsymbol{w}^b_t = \boldsymbol{w}^b_{t-1} - \frac{\eta}{\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon}\hat{\boldsymbol{m}}_t. \tag{6.13}$$

The STE method has been used as the workhorse in training many interesting extensions since the pioneering work of Courbariaux et al. (2015) and Hubara et al. (2016). Some of these works focus on adding extra scalars and parameters (Lin et al., 2017; Rastegari et al., 2016), different quantisation schemes (Wan et al., 2018; Zhu et al., 2017; Li et al., 2016), and architectures designed for quantised weights (Liu et al., 2018b).

Recently, the STE method has been used to good effect by works such as Liu et al. (2020), which have by clever architectural design pushed the performance of FBNN, and offer state-of-the-art results. Of special note are the works that removed the need for real valued weights during training (Zhou et al., 2016; Deng et al., 2018; Wang et al., 2018). These works present methods suited to training

on low compute devices. However, in this thesis we choose to focus on the more standard assumption that extra resources are available at training time, and the goal is to develop a lightweight model for inference.

6.5.5 Mirror Descent

Ajanthan et al. (2021) introduced a new method of training BNNs using Mirror Descent. Mirror Descent is a well studied first-order optimisation method for constrained convex problems (Nemirovsky et al., 1983). However, its application to the training of BNNs had not been explored previously. In particular, Ajanthan et al. (2021) introduce a numerically stable implementation of Mirror Descent that shares a lot of similarity with the STE method. For convenience, we will refer to this method as binary mirror descent (BMD). BMD also introduces an auxiliary set of parameters $\tilde{\boldsymbol{w}}^b$. However, in BMD both sets of parameters $\tilde{\boldsymbol{w}}^b$ and $\boldsymbol{w}^b$ take real values throughout training. Similar to the STE method, $\tilde{\boldsymbol{w}}^b$ is mapped to $\boldsymbol{w}^b$ before every forward pass, however here the following relation is used:

$$\boldsymbol{w}_t^b = \tanh(\beta_t \tilde{\boldsymbol{w}}_t^b), \tag{6.14}$$

where β_t is a temperature parameter. The gradient is evaluated at $\boldsymbol{w}_t^b$, and then equations (6.6) and (6.7) are used to update $\tilde{\boldsymbol{w}}_t^b$. It is worth noting that as $\beta_t \to \infty$, this mapping becomes equivalent to the sign function, and in this case BMD is identical to the STE method. However, in the BMD method, β_t starts at a low value $\beta_0 \approx 1$ and is increased throughout training according to the following scheme $\beta_t = \beta_{t-1}\lambda_{rate}$. Mirror descent is a well studied algorithm, and is more principled than the STE method which lacks strong justification from theory.

6.5.6 ProxQuant

Bai et al. (2019) introduced ProxQuant, a surprisingly simple and effective algorithm

for training neural networks with binary weights. ProxQuant is notably different from the earlier STE method and its derivatives. The ProxQuant algorithm does not require an additional auxiliary set of parameters, instead it acts on a single set of real valued weights that are initialised to a pre-trained floating point network. Throughout the training procedure $\boldsymbol{w}^b$ is slowly encouraged to become binary by use of a regularisation function $R(\boldsymbol{w})$. The regularisation function suggested by Bai et al. (2019) penalises either the l_1 or l_2 norm of the distance between $\boldsymbol{w}^b$ and its quantised value. For a binary quantisation scheme these can be detailed as follows:

$$R(\boldsymbol{w}) = \|\boldsymbol{w}^b - \text{sign}(\boldsymbol{w}^b)\|^2, \tag{6.15}$$

$$R(\boldsymbol{w}) = |\boldsymbol{w}^b - \text{sign}(\boldsymbol{w}^b)|. \tag{6.16}$$

From either of these regularisation functions a proximal operator can be derived:

$$\text{prox}_{\lambda R}(\boldsymbol{w}_t) = \underset{\boldsymbol{w}}{\text{argmin}} \left\{ \lambda R(\boldsymbol{w}) + \|\boldsymbol{w} - \boldsymbol{w}_t\|_2^2 \right\}. \tag{6.17}$$

With the proximal operator defined, the ProxQuant algorithm at time t can be summarised as:

$$\tilde{\boldsymbol{w}}_{t+1}^b = \text{prox}_{\eta_t \lambda_t R} \left(\tilde{\boldsymbol{w}}_t^b - \eta_t \nabla \ell_{z_t}(\boldsymbol{w}_t^b) \right) \tag{6.18}$$

However, Bai et al. (2019) suggest, in practice, that the Adam update (3.12) is used within the proximal operator. The regularisation amount λ_t is increased throughout training according to a linear scheme, in order to slowly push $\boldsymbol{w}^b$ to take binary values. To ensure all weights are binary in the final model for the last section of training $\boldsymbol{w}^b$ is projected onto the set $\{-1, 1\}^p$ and then $\boldsymbol{w}^r$ is fine-tuned, with the values of $\boldsymbol{w}^b$ fixed to their final values.

ProxQuant provides an alternative method to the STE method and its derivatives which is simple and produces binary models with strong generalisation performance. However, Bai et al. (2019) only consider networks with floating point activations

and do not suggest FBNN as an extension. Indeed, if trying to apply a continuation method such as this to train FBNN it is not obvious if one should binarise both weights and activations simultaneously or sequentially. Additionally, the authors only provide results for small models trained on relatively simple data sets; there is no indication if ProxQuant scales well to large models trained on data sets with more realistic image sizes.

Chapter 7

Training Binary Neural Networks the Easy Way (BNEW)

7.1 Introduction

In this chapter we present a simple but effective method for training fully binary neural networks (FBNN). Specifically, models where the majority of weights and activations are constrained to the set $\{-1, 1\}$. These models offer significant improvements in memory efficiency, energy usage and inference speed over their floating point counterparts. Our approach to training FBNN splits the task into two phases. In the first phase, a model with binary activations and floating point weights is trained. In the second, a concave regulariser is added to encourage the weights to become binary. This work is orthogonal to improvements of FBNN architectures, and offers an alternative optimisation scheme to the existing binary network optimisation schemes. Our method doesn't require an auxiliary set of weights during training and can be easily applied to any architecture without modification. Finally, we get strong results on ImageNet almost matching the state of the art.

7.2 Algorithm

7.2.1 Problem Formulation

We do not try to optimise ($\mathcal{B}$) directly and instead relax the constraint $\boldsymbol{w}^b \in \{-1, 1\}^p$ to $\boldsymbol{w}^b \in [-1, 1]^p$. In order to ensure quantised solutions we introduce a concave regularisation function $R_{BNEW}(\boldsymbol{w})$ resulting in the formulation:

$$\boldsymbol{w}_\star \in \operatorname*{argmin}_{\boldsymbol{w}\in\Omega} F_\lambda \triangleq f(\boldsymbol{w}) + \lambda R_{BNEW}(\boldsymbol{w}), \tag{$\mathcal{P}_{bin}$}$$

where $\Omega \triangleq [-1, 1]^p \cup \mathbb{R}^{d-p}$ and $R_{BNEW}(\boldsymbol{w}) \triangleq -\|\boldsymbol{w}^b\|^2 + p$. Here the addition of p ensures the non negativity of the objective, however, we will suppress this term for clarity. It is clear that as $\lambda_t \to \infty$ any solution to the above problem must have $\boldsymbol{w}^b \in \{-1, 1\}^p$, as all other solutions would incur infinite cost. With ($\mathcal{P}_{bin}$) in this form we can make use of a projected stochastic gradient descent (PSGD) update, which we detail in Section 7.2.2.

7.2.2 Parameter Update

In order to find a solution to ($\mathcal{P}_{bin}$) at time step t we solve the following proximal problem:

$$\boldsymbol{w}_{t+1} = \operatorname*{argmin}_{\boldsymbol{w}\in\Omega} \big\{ \frac{1}{2\eta_t}\|\boldsymbol{w} - \boldsymbol{w}_t\|^2 + \ell_{z_t}(\boldsymbol{w}_t) + \nabla\ell_{z_t}(\boldsymbol{w}_t)^\top(\boldsymbol{w} - \boldsymbol{w}_t) - \lambda_t\|\boldsymbol{w}^b\|^2 \big\}. \tag{7.1}$$

As (7.1) is smooth and Ω is a convex set we can simply solve for $\boldsymbol{w}_{t+1}$ to get the following update for $\boldsymbol{w}^b$:

$$\boldsymbol{w}^b_{t+1} = \Pi_{[-1,1]}\left(\frac{1}{1 - 2\lambda_t\eta_t}(\boldsymbol{w}^b_t - \eta_t\nabla\ell_{z_t}(\boldsymbol{w}^b_t))\right), \tag{7.2}$$

$$\approx \Pi_{[-1,1]}\left((1 + 2\lambda_t\eta_t)(\boldsymbol{w}^b_t - \eta_t\nabla\ell_{z_t}(\boldsymbol{w}^b_t))\right), \tag{7.3}$$

where the second equality is approximately equal for small λ_t. A derivation is provided in Appendix E. The real value parameters $\boldsymbol{w}^r$ are updated according to (6.7). Note, when $\lambda_t = 0$, (7.3) is identical to PSGD. In practice the Adam update is used over the SGD update shown inside of (6.7) and (7.3). This is done as we find it produces better empirical results. In order to find a good solution to $(\mathcal{P}_{bin})$ we used the following training process.

7.2.3 Training Procedure

Our training procedure is split into three phases. In the first phase we train a neural network with binary activations and real valued weights. In the second phase of training we slowly encourage the weights to become binary. Finally, we project $\boldsymbol{w}^b$ onto the set $\{-1, 1\}^p$, and fine-tune the real valued parameters. At a high level our training process is similar to that of ProxQuant and enjoys the same asymptotic convergence rate, see Section 7.3.

Pretraining Phase. Similar to the work of Liu et al. (2018a, 2020) we first train a neural network with binary activations and real valued weights. This is achieved by the addition of sign activations before each convolution layer. This results in the input to intermediate convolutional layers being binary. Our pretraining phase is different from that of ProxQuant in the following three ways. First, as we are concerned with training models with binary activations, the model trained during the pretraining phase has binary activations. Second, we use the Adam optimiser rather than SGD during the pretraining phase. Third, we use PSGD or (7.3) with $\lambda_t = 0$, in other words we project $\boldsymbol{w}^b$ to the set $[-1, 1]^p$ after every iteration. These last two changes address an issue with ProxQuant where the accuracy drops a lot during the first few iterations of the quantisation phase. This reduction is caused as a lot of the useful structure in the network is destroyed by $\boldsymbol{w}^b$ being suddenly clipped to $[-1, 1]^p$ and a jump to a much higher effective learning rate for some

parameters, as a result of switching from SGD to Adam.

Quantisation Phase. Once the first phase of training is complete we proceed with the quantisation phase. In this phase we slowly move from a model with both real valued weights and binary activations to a model with binary weights and activations. We start from the best model from pretraining and increase the weight of the regularisation function λ_t throughout this phase, according to a linear rate $\lambda_t = t \cdot \lambda_{rate}$. In general, completing the quantisation phase over a longer time frame with a lower value of λ_{rate} produces better results.

Finetuning. Before the last few epochs of training we project $\boldsymbol{w}^b$ onto the set of quantised values $\{-1, 1\}^p$. We then set $\nabla \ell_{z_t}(\boldsymbol{w}_t^b) = 0$. This process is done in order to ensure all weights are binary and to fine-tune the real value parameters $\boldsymbol{w}^r$ to the final values of $\boldsymbol{w}^b$.

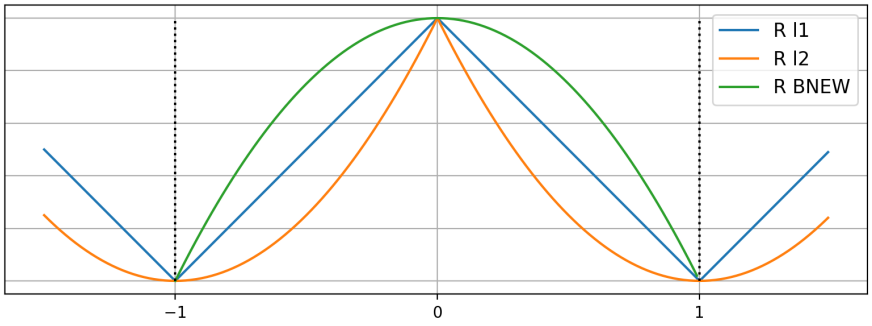


Figure 7.1: *Different choices for regularisation functions penalising non-binary weights.*

7.2.4 Choice of Regularisation Function

Bai et al. (2019) present two different choices for regularisation functions, detailed in equation (6.15). While these are natural choices, and produce reasonable results, both functions have their largest gradients close to zero. Hence, as λ_t is increased the

weights close to zero are quickly pushed in either the positive or negative direction. Moreover, once $\mathbb{E}_t[\nabla \ell_{z_t}(w_t^b)] \leq \mathbb{E}_t[\lambda_t \nabla R(w_t^b)]$ it becomes difficult for w^b to change sign for the rest of training. Figure 7.1 depicts our choice of regularisation function $R_{BNEW}(\boldsymbol{w}) \triangleq -\|\boldsymbol{w}^b\|^2 + p$ and those introduced in Bai et al. (2019). We suggest R_{BNEW} is a more appropriate choice for the following reasons. First, R_{BNEW} has small gradient close to zero. This means parameters close to zero do not experience as large an update towards -1 or $+1$ as parameters already close to these values. Hence, in general the parameters close to their quantised values are more incentivized to take binary values first, then followed by those with smaller absolute value. The large gradient of R_{BNEW} close to -1 or $+1$ is designed to help prevent parameters oscillating around these values. Finally, and most importantly we find that our R_{BNEW} produces better results in practice.

7.3 Theoretical Justification

BNEW enjoys the same theoretical convergence rate as ProxQuant. Here we restate the result presented in Bai et al. (2019). We note that the following rate assumes that f is smooth which is not the case when including ReLU or sign functions within the network. However, as suggested in Bai et al. (2019) it is easy to use smoothed alternatives to these functions. For example, $\tanh(k\boldsymbol{x})$ with an appropriate choice of k can be used in place of $\text{sign}(\boldsymbol{x})$ to get a desired level of smoothness.

Theorem 4 (BNEW). *We assume that f is β-smooth. Let $F_* \triangleq \min_\Omega F_\lambda(\boldsymbol{w})$. We further assume that $\eta_t = \frac{1}{2\beta}$, $\forall t$ and we have access to the batch gradient ∇f and $\lambda_t = \lambda$ then if we use BNEW with updates (6.7) and (7.3) for T steps we have:*

$$\|\nabla F_\lambda(\boldsymbol{w}_{T_{best}})\|^2 \leq \frac{C\beta(F_\lambda(\boldsymbol{w}_0) - F_*)}{T}, \tag{7.4}$$

where $C > 0$ is a constant and T_{best} is defined as $T_{best} \triangleq \text{argmin}_{1 \leq t \leq T} \|\boldsymbol{w}_t - \boldsymbol{w}_{t-1}\|$.

7.4 Experiments

Our experiments are split into two sections. In the first we investigate the performance of BNEW using a small FBNN on CIFAR-100 (Krizhevsky, 2009). In the second section we show that BNEW scales to large FBNN and data sets by training a ReActNet (Liu et al., 2020) on the ImageNet data set (Deng et al., 2009).

7.4.1 Small Scale Experiments

Setting. Many of the previous works on FBNNs only present results for large over parameterised models (Ajanthan et al., 2021; Liu et al., 2020, 2018a). While these models may seem appealing due to the smaller accuracy degradation from their real valued counterparts, smaller FBNN architectures are preferable to run on embedded devices. In this section we provide results by training a small ResNet20 (He et al., 2016), on the CIFAR-100 data set (Krizhevsky, 2009). This data set contains 60,000, 32x32 pixel RGB images split over 100 classes. We modified a ResNet20 to include a similar block structure to ReActNets, detailed in Liu et al. (2020). This results in a small effective FBNN with $d = 0.28$M. We let the weights in the first and last layers of the network, all biases, parameters of batch norm layers, and bottleneck layers retain floating point values. This results in a model which is 95% binary and requires 12.5 times less memory to store compared to its floating point counterpart. The speed of this model would depend on the exact hardware used at inference time. However, due to its similarity in design to models suggested by Rastegari et al. (2016) a speed increase of up to 50 times is likely when using this FBNN, over a real valued ResNet20 on CPU.

Method: Here we compare BNEW against the STE method as described in Section 6.5.4, and BMD described in Section 6.5.5. We select these methods as they are optimisation algorithms that can be used to train any FBNN. Importantly they do not require any modification of the network architecture, which sets them apart

from many works on FBNN.

For a fair comparison we use the following hyperparameters and strategies for all methods. We use a batch size of 128. We present results for quantisation phases with two different epoch budgets, specifically 200 and 1000. We use Adam with a linearly decaying learning rate schedule with $\eta_0 \in \{0.01, 0, 001\}$. For BMD and BNEW we cross-validate $\lambda_{rate} \in \{1.003, 1.01, 1.03, 1.1, 1.3\}$ and $\lambda_{rate} = \{0.001, 0.0001\}$, respectively. To ensure the $\boldsymbol{w}^b$ has binary values for both of these methods the real valued weights are finetuned for the final 20 epochs, as described in Section 7.2.3. We include results for a baseline ResNet20 with real valued weights and activations for use as a reference, trained according to the scheme described in He et al. (2016). All results are computed over five runs with different random seeds. We use the same pre-trained models with binary activations and real valued parameters as the starting point for all methods as described in Section 7.2.3. While the authors of BMD do not suggest this for their method, we find it produces superior results. All images are centred and normalised per channel and are subject to random flips and crops during training.

Specifically we replace the cross entropy loss with a loss that minimises the Kullback–Leibler divergence between a real valued teacher model and the binary student. This is done during both the pretraining and quantisation phases.

Results: The results of the CIFAR-100 experiments are shown in Table 7.1. For all methods the longer quantisation phase, and use of distillation produces superior results. Thus if the computation budget allows, training for longer and utilising distillation should be encouraged to produce a stronger binary model. BNEW saw the largest gain in accuracy when using a higher epoch training budget. For the 1000 epoch budget BNEW produced the best binary model outperforming the alternatives by 0.7%, and thus resulting in an accuracy degradation of less than 10% over its floating point counterpart. For the 200 epoch budget the STE method was the most

Table 7.1: *Resulting accuracies when training a modified FBNN ResNet20 on the CIFAR-100 data set with different FBNN training schemes.*

Training Epochs:		200	1000
Optimiser	Distillation	Test Accuracy (%)	
Real Valued	No	67.1σ0.7	-
STE	No	**53.6σ0.9**	**55.0σ0.5**
BMD	No	53.3σ0.4	54.8σ0.5
BNEW	No	52.7σ0.3	**55.0σ0.3**
STE	Yes	**56.1σ0.4**	56.8σ0.3
BMD	Yes	55.7σ0.6	56.8σ0.5
BNEW	Yes	55.8σ 0.5	**57.5σ0.3**

successful, and we would recommend its use when the training budget is limited, however this is not the typical paradigm for FBNN.

7.5 Ablation Study

Setting. In this section we investigate the performance when removing various aspects of the ReActNet architectures (Liu et al., 2020). ReActNet is a bespoke fully binary neural network architecture, which achieves state-of-the-art performance on ImageNet. ReActNet is based on MobileNet (Howard et al., 2019) with a number of modifications making it better suited for use with binary weights and activations. In Section 7.4.1 we provided results of training a ResNet20 (He et al., 2016) with these modifications using STE, BMD and BNEW. Here, using the same model we investigate the effect of removing these modifications one at a time on the performance of BNEW. We also provide results produced using the STE method for a comparison. We do this to help disentangle what architectural choices are useful irrespective of optimisation method.

Method. Similar to Section 7.4.1 we use a 200 epoch budget and $\eta_0 = 0.01$ in combination with a linearly decaying step size schedule. We use $\eta_0 = 0.01$ with a linear

decay for the pretraining and quantisation phases. We use $\lambda_{rate} = 0.01$. We again report test error and standard deviation values calculated over five independent runs with different random seeds.

Baseline. The baseline model that we perform the Ablation study on is the model detailed in Section 7.4.1 trained with distillation.

Learnable Bias Layers. Liu et al. (2020) increase the expressive power of ReActNet by using RSign and RPReLU activation functions rather than the non-parametric versions; Sign and PReLU. RSign and RPReLU are generalised activation functions, which effectively add additional real valued parameters to each channel in the form of a bias, see Liu et al. (2020) for more details. This modification can be viewed as adding several learnable bias layers to the model. These extra layers are located before each sign activation and before and after each PReLU. However, as these biases are per channel, in practice they only increase the size and computational cost of the network marginally.

PReLU vs ReLU. To quantify the benefit of the PReLU non-linearities we train an additional model containing the learnable bias layers in combination with ReLU activations instead of the PReLU activations.

Parameter Scaling. ReActNet uses a binary quantisation scheme where the binary parameters are scaled per channel, specifically with $w \in \{-\alpha_c, \alpha_c\}$ where c indexes over the output channels of a given layer. The scalars α_c are calculated to be the mean of the absolute values of the parameters in the c^{th} output layer. Note, once a model is trained, that parameters $\boldsymbol{w}^b$ can then be converted to $\{-1, 1\}^p$ by multiplying the relevant batch-norm parameters by α_c. We employ a similar scaling, but we use a learnable scale per channel, as this leads to an easier comparison. However, we note in both cases, due to the presence of batch norm, the inclusion of

the scalars should have little to no effect.

Distillation. Similar to Section 7.4.1 we detail the performance without distillation.

Choice of Approximation for Sign Function's Gradient. In equations (6.2) and (6.3) we detail two choices to approximate the gradient of the sign function. In Section 7.4.1 we made use of the more complex version (6.3), here we investigate the effect of instead using the original Straight Through Estimator (equation (6.2)) as suggested in Hubara et al. (2016). We label the model with this modification "Classic STE".

Binary First and Last Layers. It is standard to retain floating point parameters within the first and last layer of an FBNN. We investigate the effect of making these layers binary as well.

Binary Bottlenecks. A number of recent works (Bethge et al., 2019; Liu et al., 2020) recommend against binary 1x1 convolutional layers in bottlenecks. We investigate the effect on performance of ignoring this advice, and quantising these layers as well.

No Pretraining. In order to determine how important the pretraining phase is to the accuracy we try skipping this phase. We instead run the quantisation phase directly on a random initialisation.

Different Regularisation functions. Finally we include results for BNEW using the regularisation functions detailed in (6.15).

7.5.1 Results.

The results of the ablation study are shown in Table 7.2. Out of all the modifications considered here, binarising the first and last layer caused the largest accuracy degradation of over 10%. Binarising the bottleneck layers resulted in the second largest drop of 4%, reaffirming the suggestion of Bethge et al. (2019) that binary bottleneck layers should be avoided. Removing distillation resulted in the third largest drop in accuracy at a more modest 3%. Skipping the per-training phase and training the model from scratch resulted in a performance loss of roughly 2%. Using ReLU activations but still including the bias layers resulted in a 1.3% drop. Not using the learnable bias layers only resulted in a 0.2% drop in accuracy suggesting that in this setting these floating point weights could be excluded with minor cost, resulting in even faster inference. We found in this study that the classic STE performed better than the more complex approximation of the sign function, equation (2), introduced by Liu et al. (2018b). However, the difference here is not statistically significant, and we would suggest trying both versions as there does not seem to be a consensus in the literature on which works better (Liu et al., 2018b; Bethge et al., 2019).

The results of this ablation study suggest that the performance of a FBNN architecture is insensitive to the training method used. The differences in the changes in performance between the two methods were relatively consistent, with STE slightly outperforming BNEW due to the small epoch budget used.

As a result of the ablation study we repeat the experiments from Section 7.4.1 with a model that uses the Classic STE to approximate the gradient of the sign function and does not include weight scaling parameters. Distillation was used again. We present the results of this experiment in Table 7.3. These changes boost performance in general, suggesting that these aspects of the ReActNet architecture are not always necessary. For this model, all optimisation schemes work nearly identically for both settings, the exception being BMD which performed worse for the 1000 epoch budget. We next investigate how BNEW scales by training a much

larger binary model on the ImageNet data set.

Table 7.2: *Ablation study test accuracies.*

Real Values	67.1σ0.7	
Optimiser	STE	BNEW
Baseline	56.1σ0.4	55.8σ0.5
No Learnable Bias Layers	55.6σ0.5	55.6σ0.6
No PReLU	54.3σ0.1	54.5σ0.3
No Scale	56.2σ0.3	56.0σ0.5
No Distillation	53.6σ0.9	52.7σ0.5
Classic STE	56.5σ0.3	56.0σ0.4
Binary First and Last	43.5σ1.2	42.5σ0.5
Binary Bottlenecks	52.9σ0.3	51.8σ0.3
No Pretrain	55.0σ0.2	54.0σ0.5
R_{ℓ_1} - Regulariser	NA	54.5σ0.5
R_{ℓ_2} - Regulariser	NA	54.9σ0.4

Table 7.3: *Accuracies of modified mini-ReActNet on CIFAR-100 data set.*

Training Epochs:	200	1000
Optimiser	Test Accuracy (%)	
Real Valued	67.1σ0.7	
STE	**56.2σ0.4**	57.3σ0.4
BMD	56.0σ0.4	56.9σ0.4
BNEW	56.0σ0.3	**57.4σ0.4**

7.5.2 ImageNet Experiments

Setting. The ImageNet data set is a large data set with more realistic image sizes than CIFAR. Thus, it gives a indication of what sort of performance is possible when using an FBNN for a real world image classification application. We give more details of this data set in Section 5.4.3.

Method. On this data set we train a ReActNet-A (Liu et al., 2020), and use their data augmentation scheme. The ground truth labels for ImageNet are not freely

available and hence we report validation scores instead. We follow the training methodology of Liu et al. (2020), however in phase two of training we use BNEW rather then the STE to convert the parameters within $\boldsymbol{w}^b$ from real values to binary values. We use $\lambda_{rate} = 0.01$ $\eta_0 = 10^{-3}, \text{epochs} = 500, \text{epoch}_{freeze} = 400$. We restart the linear decay of the η for the finetuning phase. We do not change any other hyperparameters except the batch size which we reduce to 220 due to hardware constraints. We compare against other FBNN models trained on ImageNet with comparable compute budgets as calculated by Liu et al. (2020). Specifically we compare against, XNORnet (Rastegari et al., 2016), BiRealNet (Liu et al., 2018a), BMD (Ajanthan et al., 2021), Real-to-Bin (Martinez et al., 2020), BOP (Helwegen et al., 2019) 2^{nd} Order BOP (2OB) (Suarez-Ramirez et al., 2021) and ReActNet (Liu et al., 2020).

Table 7.4: *Accuracies of different FBNN of comparable computational budget evaluated on the ImageNet data set.*

Architecture	Distillation	Backbone	Optimiser	Accuracy
XNORnet	No	ResNet18	STE	51.2
BiRealNet	No	ResNet18	STE	56.4
ReActNet	No	ResNet18	BOP	56.6
ReActNet	No	ResNet18	2OB	57.2
ReActNet	Yes	ResNet18	STE	65.5
Real-to-Bin	Yes	ResNet18	STE	65.4
ReActNet	Yes	ResNet18	STE	65.5
ReActNet	Yes	MobileNet	STE	69.4
ReActNet	Yes	MobileNet	BNEW	**69.7**

Results. BNEW achieves an accuracy of 69.7% which is 0.3% higher than the state-of-the-art performance of Liu et al. (2020). This is an especially encouraging result due to the fact that their network architecture was designed for use with the STE method. This shows our simple approach for training FBNN easily scales to large models and produces strong results. BNEW is particularly well suited to train

very large BNN, that do not fit on a single GPU when trained with the STE or BMD methods due to its reduced memory requirement.

7.6 Discussion

In this chapter we have introduced BNEW; an effective and simple method for training neural networks with both binary weights and activations. To our knowledge, this is the first demonstration that a simple continuation method is effective for models of this type. BNEW's simplicity makes it easily applicable in a wide range of settings. Additionally, it has the following advantageous qualities over the STE and BMD methods: i) better or comparable empirical results ; ii) does not require additional parameters during training; and iii) a clear objective that is being minimised and a strong theoretical justification. BNEW also has two main weaknesses. First the full training procedure must be completed up to the fine tuning phase before one has any indication of the final performance. Second, BNEW's effectiveness has a dependence on the hyperparamater λ_{rate} which needs to be selected to ensure $\boldsymbol{w}^b$ becomes mostly binary before the finetuning phase. However, this issue can be circumvented by continuing training until this is so. In this work we have only shown results for binary quantisation, however, it would be trivial to extend BNEW to other quantisation schemes by modifying the regularisation function R_{BNEW}.

Chapter 8

Conclusion

8.1 Summary

In this thesis, we present three optimisation algorithms for training deep neural networks in different settings.

In Chapter 4, we showed how a bundle approximation can be used for training interpolating networks which we named Bundle Optimiser for Robust and Accurate Training (BORAT). We showed how to combine N steps of a gradient based optimiser to form a single proximal update. BORAT has a comparable run time to the existing ALI-G optimiser, while reducing the sensitivity of the optimisation to the learning rate, regularisation parameter, and label noise. This results in an optimisation algorithm that requires less tuning, and thus reduces the expected time and energy needed to train a high accuracy model.

In Chapter 5, we presented a second extension to ALI-G that allows it to be easily applied to the non-interpolating setting without adding any extra hyperparameters. We named this extension ALI-G+, as it increases the number of settings in which ALI-G can be applied. ALI-G+ uses a simple heuristic to estimate the optimal loss value per sample online. This results in an easy to use optimiser that does not require a learning rate schedule. ALI-G+ performs favourably against other single

hyperparameter optimisation techniques, both in the non-interpolating and many interpolating settings.

In Chapter 7, we shifted our focus to training deep neural networks with both binary weights and activations. Here, we proposed Binary Networks the Easy Way (BNEW) which offers a simple but effective alternative method for training fully binary models. This method simply relaxes the constraint set to its convex hull and introduces a concave regularisation function. This function is applied to the subset of the network parameters it is desired to quantise. The method starts by training a network with binary activations and real valued parameters. Then the strength of the regularisation is increased to encourage the chosen parameters to become binary. BNEW has two main advantages. First, it has a smaller memory footprint during training, as it does not require auxiliary parameters. Second, BNEW offers strong empirical results when given a sufficient epoch budget.

These three optimisation algorithms are designed to be easy to use and reduce the energy requirements of neural networks in different settings. BORAT and ALI-G+ aim to minimise the energy needed in finding a high accuracy model in the interpolating and non-interpolating settings, respectively. Conversely, BNEW aims to reduce the energy cost at inference time by facilitating the easy training of highly accurate binary models that can be run on less powerful hardware or with cheaper hardware instructions. Although the power of scaling up neural networks has led to some very spectacular results Saharia et al. (2022); Brown et al. (2020); Shoeybi et al. (2019); Devlin et al. (2019), the increasing cost of these models, both in terms of money and energy demands, is less well publicised. While less sensational investigation into methods to reduce these costs is equally important and necessary of further work. To this end, in the following section we explore a few directions for research that we believe to be important to help achieve this goal.

8.1.1 Future Work

In this section we highlight a few areas of research we believe would have high impact in improving the efficiency of deploying neural networks.

The first direction is further research to improve the efficiency of training neural networks. While BORAT and ALI-G+ both produce strong results with a single fixed hyperparameter they do not yet outperform SGD with a well tuned schedule. As a result of this, SGD with a large cross-validation scheme is common in order to maximise network performance. This is particularly prevalent in machine learning research itself, where generalisation performance is often used as the only performance metric. Thus, extensively refining hyperparameters to boost performance is highly incentivised. Sivaprasad et al. (2020) introduce the idea of training efficiency, and state optimiser performance should take account for hyperparameter tuning; however these idea yet to see widespread adoption.

Outside the field of machine learning research, foundation models (Bommasani et al., 2021) offer the promise of reduced training costs. "Foundation Models" is a recent name given to very large models trained on extremely large data sets. Foundation Models are typically transformers (Vaswani et al., 2017) and trained in an unsupervised or semi-unsupervised fashion. These models demonstrate the ability to be very easily converted to new tasks, often by simply adding a new final layer on top of the fixed parameters of the base model. This layer can even be found by ridge regression and can offer good performance on new tasks even with very limited training data (Galanti et al., 2021). The main issue with foundation models is their vast size, many having billions of parameters (Yuan et al., 2021). Thus, while finding a new model with high accuracy is relatively cost efficient in terms of compute, these savings are quickly lost at inference time. Foundation models also require a large amount of memory due to their size. It remains to be seen to what level foundation models can be compressed without losing their strong generalisation and transfer learning performance. However recently, some works are

starting to tackle this question (Du et al., 2021).

This leads into another area we believe is in need of further research, that is, models with low memory and inference costs. We believe the largest outstanding question here is what form of network compression is most effective? Specifically, when aiming to produce a model with a set size, inference budget and training budget, which network compression scheme results in the model with the best generalisation performance? These methods have been compared qualitatively but to the best of our knowledge it is an open question which method is currently the most cost effective (Neill, 2020). We speculate that this is because a fair quantitative comparison of these techniques is difficult, especially when considering a wide range of tasks and scales.

Finally, it is worth mentioning research directly into the improvement of hardware efficiency, as this would have a large impact on the environmental cost of deploying neural networks. While not normally categorised as machine learning research, reducing the run-time energy, manufacturing, and end of life costs of hardware such as GPUs and TPUs, typically used to deploy neural networks, would be highly beneficial.

In the coming decade, machine learning has the potential to benefit humanity in a number of ways. However, it is essential that its impact on the environment is minimised so its contribution to global climate change does not undermine these benefits.

Appendix A

Appendix: Proofs of Theorems in Chapter 4

A.1 Dual Derivation

Lemma 1. *The dual of the primal problem (4.3), can be written as follows:*

$$\sup_{\boldsymbol{\alpha}\in\Delta_N}\left\{-\frac{\eta}{2}\boldsymbol{\alpha}^\top A_t^\top A_t\boldsymbol{\alpha}+\boldsymbol{\alpha}^\top \boldsymbol{b}_t^n\right\}.$$

Where A_t is defined as a $d\times N$ matrix, whose n^{th} row is $\nabla\ell_{z_t^n}(\hat{\boldsymbol{w}}_t^n)$, $\boldsymbol{b}_t^n=[b_t^1,...,b_t^N]^\top$ and $\boldsymbol{\alpha}=[\alpha^1,\alpha^2,\ldots,\alpha^N]^\top$.

Proof. We start from the primal problem:

$$\operatorname*{argmin}_{\boldsymbol{w}\in\mathbb{R}^d}\left\{\frac{1}{2\eta}||\boldsymbol{w}-\boldsymbol{w}_t||^2+\max_{n\in[N]}\left\{\nabla\ell_{z_t^n}(\hat{\boldsymbol{w}}_t^n)^\top(\boldsymbol{w}-\boldsymbol{w}_t)+b_t^n\right\}\right\}, \tag{A.1}$$

We define $\tilde{\boldsymbol{w}} = \boldsymbol{w} - \boldsymbol{w}_t$.

$$
\begin{aligned}
&\min_{\boldsymbol{w}\in\mathbb{R}^d} \left\{ \frac{1}{2\eta}||\tilde{\boldsymbol{w}}||^2 + \max_{n\in[N]} \left\{ \nabla \ell_{z_t^n}(\hat{\boldsymbol{w}}_t^n)^\top \tilde{\boldsymbol{w}} + b_t^n \right\} \right\}, \\
&\min_{\boldsymbol{w}\in\mathbb{R}^d,\zeta} \left\{ \frac{1}{2\eta}||\tilde{\boldsymbol{w}}||^2 + \zeta \right\} \text{ subject to: } \zeta \geq \nabla \ell_{z_t^n}(\hat{\boldsymbol{w}}_t^n)^\top \tilde{\boldsymbol{w}} + b_t^n, \ \forall n \in [N], \\
&\min_{\tilde{\boldsymbol{w}}\in\mathbb{R}^d,\zeta} \sup_{\alpha^n\geq 0,\forall n} \left\{ \frac{1}{2\eta}||\tilde{\boldsymbol{w}}||^2 + \zeta - \sum_{n=1}^N \alpha^n (\zeta - \nabla \ell_{z_t^n}(\hat{\boldsymbol{w}}_t^n)^\top \tilde{\boldsymbol{w}} - b_t^n) \right\}, \\
&\sup_{\alpha^n\geq 0,\forall n} \min_{\tilde{\boldsymbol{w}}\in\mathbb{R}^d,\zeta} \left\{ \frac{1}{2\eta}||\tilde{\boldsymbol{w}}||^2 + \zeta - \sum_{n=1}^N \alpha^n (\zeta - \nabla \ell_{z_t^n}(\hat{\boldsymbol{w}}_t^n)^\top \tilde{\boldsymbol{w}} - b_t^n) \right\}, \quad \text{(strong duality)} \\
&\sup_{\alpha^n\geq 0,\forall n} \min_{\tilde{\boldsymbol{w}}\in\mathbb{R}^d,\zeta} \left\{ \frac{1}{2\eta}||\tilde{\boldsymbol{w}}||^2 + \zeta + \sum_{n=1}^N \alpha^n (\nabla \ell_{z_t^n}(\hat{\boldsymbol{w}}_t^n)^\top \tilde{\boldsymbol{w}} + b_t^n - \zeta) \right\}.
\end{aligned}
$$

The inner problem is now smooth in $\tilde{\boldsymbol{w}}$ and ζ. We write the KKT conditions:

$$
\begin{aligned}
\frac{\partial \cdot}{\partial \tilde{\boldsymbol{w}}} &= \frac{\tilde{\boldsymbol{w}}}{\eta} + \sum_{n=1}^N \alpha^n \nabla \ell_{z_t^n}(\hat{\boldsymbol{w}}_t^n) = 0 \\
\frac{\partial \cdot}{\partial \zeta} &= 1 - \sum_{n=1}^N \alpha^n = 0
\end{aligned}
$$

We define $\Delta_N \triangleq \{\boldsymbol{\alpha} \in \mathbb{R}^N : \sum_{n=1}^N \alpha^n = 1, \alpha^n \geq 0, n = 1, ..., N\}$ as a probability simplex over the elements of $\boldsymbol{\alpha}$. Thus, when we plug in the KKT conditions we obtain:

$$
\begin{aligned}
&\sup_{\boldsymbol{\alpha}\in\Delta_N} \left\{ \frac{1}{2\eta}||\eta \sum_{n=1}^N \alpha^n \nabla \ell_{z_t^n}(\hat{\boldsymbol{w}}_t^n)||^2 + \sum_{n=1}^N \alpha^n \left(-\nabla \ell_{z_t^n}(\hat{\boldsymbol{w}}_t^n)^\top (\eta \sum_{m=1}^N \alpha^m \nabla \ell_{z_t^m}(\hat{\boldsymbol{w}}_t^m)) + b_t^n \right) \right\}, \\
&\sup_{\boldsymbol{\alpha}\in\Delta_N} \left\{ \frac{\eta}{2}||\sum_{n=1}^N \alpha^n \nabla \ell_{z_t^n}(\hat{\boldsymbol{w}}_t^n)||^2 - \eta \sum_{n=1}^N \alpha^n (\nabla \ell_{z_t^n}(\hat{\boldsymbol{w}}_t^n)^\top \sum_{m=1}^N \alpha^m \nabla \ell_{z_t^m}(\hat{\boldsymbol{w}}_t^m)) + \sum_{n=1}^N \alpha^n b_t^n \right\}, \\
&\sup_{\boldsymbol{\alpha}\in\Delta_N} \left\{ \frac{\eta}{2}||\sum_{n=1}^N \alpha^n \nabla \ell_{z_t^n}(\hat{\boldsymbol{w}}_t^n)||^2 - \eta||\sum_{n=1}^N \alpha^n \nabla \ell_{z_t^n}(\hat{\boldsymbol{w}}_t^n)||^2 + \sum_{n=1}^N \alpha^n b_t^n \right\}, \\
&\sup_{\boldsymbol{\alpha}\in\Delta_N} \left\{ -\frac{\eta}{2}||\sum_{n=1}^N \alpha^n \nabla \ell_{z_t^n}(\hat{\boldsymbol{w}}_t^n)||^2 + \sum_{n=1}^N \alpha^n b_t^n \right\}.
\end{aligned}
$$

Using the definitions of A_t, $\boldsymbol{b}_t^n$ and $\boldsymbol{\alpha}$, we can recover the following compact form

for the dual problem:

$$\sup_{\boldsymbol{\alpha}\in\Delta_N} \left\{ -\frac{\eta}{2}\boldsymbol{\alpha}^\top A_t^\top A_t \boldsymbol{\alpha} + \boldsymbol{\alpha}^\top \boldsymbol{b}_t^n \right\}.$$

□

A.2 Proof of Proposition 1

Proposition 1. *Let $F : \mathbb{R}^N \to \mathbb{R}$ be a concave function. Let us define $\boldsymbol{\alpha}_* = \operatorname{argmax}_{\boldsymbol{\alpha}\in\Delta} F(\boldsymbol{\alpha})$. Then there exists $c \in \mathbb{R}$ such that:*

$$\forall n \in [N] \text{ such that } \alpha_*^n > 0, \text{ we have: } \left.\frac{\partial F(\boldsymbol{\alpha})}{\partial \alpha^n}\right|_{\boldsymbol{\alpha}=\boldsymbol{\alpha}_*} = c. \tag{A.2}$$

In other words, the value of the partial derivative is shared among all coordinates of $\boldsymbol{\alpha}_*$ that are non-zero.

Proof. We start from the optimality condition for constrained problems:

$$\langle \nabla F(\boldsymbol{\alpha}_*), \boldsymbol{\alpha}_* - \boldsymbol{\alpha} \rangle \geq 0, \quad \forall \boldsymbol{\alpha} \in \Delta \tag{A.3}$$

We consider the point $\hat{\boldsymbol{\alpha}}$. Without loss of generality we assume $\hat{\boldsymbol{\alpha}}$ is equal to $\boldsymbol{\alpha}_*$ for all but two dimensions i and j. We let $\hat{\alpha}_i = 0$ and $\hat{\alpha}_j = \alpha_i^* + \alpha_j^*$. Hence, we know $\hat{\boldsymbol{\alpha}} \in \Delta$ as we have $\sum_i \hat{\alpha}_i = \sum_i \alpha_i^* = 1$ and $\hat{\alpha}_i \geq 0$, $\forall i$. Letting $\boldsymbol{\alpha} = \hat{\boldsymbol{\alpha}}$ in (A.3) give us:

$$-\frac{\partial F(\boldsymbol{\alpha}_*)}{\partial \alpha_i}\alpha_i^* + \frac{\partial F(\boldsymbol{\alpha}_*)}{\partial \alpha_j}\alpha_i^* \geq 0. \tag{A.4}$$

Rearranging we have:

$$\frac{\partial F(\boldsymbol{\alpha}_*)}{\partial \alpha_j}\alpha_i^* \geq \frac{\partial F(\boldsymbol{\alpha}_*)}{\partial \alpha_i}\alpha_i^*. \tag{A.5}$$

Hence, for any $\alpha_i^* \neq 0$, we have:

$$\frac{\partial F(\boldsymbol{\alpha}_*)}{\partial \alpha_j} \geq \frac{\partial F(\boldsymbol{\alpha}_*)}{\partial \alpha_i}. \tag{A.6}$$

Noticing this result holds for any i and j gives us the following and proves the result.

$$\frac{\partial F(\boldsymbol{\alpha}_*)}{\partial \alpha_j} = \frac{\partial F(\boldsymbol{\alpha}_*)}{\partial \alpha_i} = c \tag{A.7}$$

□

A.3 Proof of Proposition 3

Proposition 3. *Algorithm 1 returns a solution* $\boldsymbol{\alpha}^*$ *that satisfies* $\boldsymbol{\alpha}^* \in \operatorname{argmax}_{\boldsymbol{\alpha}\in\Delta_N} D(\boldsymbol{\alpha})$. *This is true even when a the dual does not have a unique solution.*

Proof. Let $\boldsymbol{x}^* \in \operatorname{argmax}_{\boldsymbol{\alpha}\in\Delta_N} D(\boldsymbol{\alpha})$ such that $\boldsymbol{x}^*$ has a maximal number of zero coordinates. $\boldsymbol{x}^*$ exists as we know the solution set is non-empty. Let I be the set of non-zero coordinates of $\boldsymbol{x}^*$. We denote by $S^{(I)}$ the set of solutions to the linear system associated with I:

$$\mathcal{S}^{(I)} \triangleq \left\{\boldsymbol{x} \in \mathbb{R}^{|I|} : \tilde{Q}\boldsymbol{x} = \tilde{\boldsymbol{b}}, \boldsymbol{x} \geq 0\right\}, \text{where}, \tilde{Q} \triangleq \begin{bmatrix} Q_{[I\times I]} & \mathbf{1} \\ \mathbf{1}^\top & 0 \end{bmatrix} \text{and}, \tilde{\boldsymbol{b}} = \begin{bmatrix} \boldsymbol{b}_{[I]} \\ 1 \end{bmatrix}. \tag{A.8}$$

$\mathcal{S}^{(I)}$ is a polytope as it is the intersection between the probability simplex and a linear sub-space. Therefore it admits a vertex representation:

$$\mathcal{S}^{(I)} = \operatorname{Conv}(\mathcal{V}^{(I)}) \tag{A.9}$$

such that $\operatorname{Conv}(\cdot)$ denotes the convex hull operation, and $\mathcal{V}^{(I)}$ is a finite set. Since $\mathcal{S}^{(I)}$ contains $\boldsymbol{x}^*[I]$, $\mathcal{S}^{(I)}$ is non-empty and neither is $\mathcal{V}^{(I)}$. Let v be and element of $\mathcal{V}^{(I)}$. Note, that if v had one or more zero coordinates, it would be a solution after lifting to $\mathbb{R}^N$. It would also be an optimal solution to the dual as good as $\boldsymbol{x}^*$ while having more zero coordinates than x*. This is impossible by the definition of $\boldsymbol{x}^*$. Thus, we can conclude v has exclusively non-zero coordinates.

Since v is an external point of $S^{(I)}$, see section 2.1 of Bertsekas (2009) for a definition, it then follows from proposition 2.1.4b of Bertsekas (2009) that the columns of $\tilde{Q}$ are independent. Therefore, the linear system admits a unique solution $\boldsymbol{x}^*$, which is found by Algorithm 1. □

A.4 Convex Results

Lemma 2. *Adding additional linear approximations to the bundle of BORAT can never result in a lower maximal dual value. Formally:*

$$D^m(\boldsymbol{\alpha}_*) \geq D^l(\boldsymbol{\alpha}_*) \quad \forall m > l. \tag{A.10}$$

Proof. Any vector of Δ_l can be lifted to Δ_m by appending $m - l$ zeros to it, which does not impact the value of the objective function. The lifted set Δ_l is then a subset of Δ_m, hence the result (maximisation performed over a larger space). □

Theorem 5 (Convex)**.** *We assume that Ω is a convex set, and that for every $z \in \mathcal{Z}$, ℓ_z is convex. Let $\boldsymbol{w}_\star$ be a solution of ($\mathcal{P}$), and assume that we have perfect interpolation: $\forall z \in \mathcal{Z}$, $\ell_z(\boldsymbol{w}_\star) = 0$. Then BORAT for $N \geq 2$ applied to f satisfies:*

$$\|\boldsymbol{w}_{t+1} - \boldsymbol{w}^*\|^2 \leq \|\boldsymbol{w}_t - \boldsymbol{w}^*\|^2 - 2\eta \max_{\boldsymbol{\alpha} \in \Delta_N} D(\boldsymbol{\alpha}), \tag{A.11}$$

where D is defined in (4.4).

Proof. We start by plugging the parameter update into the expression for the Euclidean distance from the next iterate to the optimal point:

$$\|\boldsymbol{w}_{t+1} - \boldsymbol{w}^*\|^2 \leq \|\Pi_\Omega(\boldsymbol{w}_t - \eta A_t \boldsymbol{\alpha}_t) - \boldsymbol{w}^*\|^2, \tag{A.12}$$

$$\leq \|\boldsymbol{w}_t - \eta A_t \boldsymbol{\alpha}_t - \boldsymbol{w}^*\|^2, \qquad (\Pi_\Omega \text{ projection}) \tag{A.13}$$

$$\leq \|\boldsymbol{w}_t - \boldsymbol{w}^*\|^2 + \eta^2 \|A_t \boldsymbol{\alpha}_t\|^2 - 2\eta \langle A_t \boldsymbol{\alpha}_t, \boldsymbol{w}_t - \boldsymbol{w}^* \rangle, \qquad (\text{expanding}) \tag{A.14}$$

$$\|\boldsymbol{w}_{t+1} - \boldsymbol{w}^*\|^2 - \|\boldsymbol{w}_t - \boldsymbol{w}^*\|^2 \leq \eta^2 \|A_t \boldsymbol{\alpha}_t\|^2 - 2\eta \langle A_t \boldsymbol{\alpha}_t, \boldsymbol{w}_t - \boldsymbol{w}^* \rangle, \quad (\text{rearranging}) \tag{A.15}$$

$$= \eta^2 \|A_t \boldsymbol{\alpha}_t\|^2 - 2\eta \langle A_t \boldsymbol{\alpha}_t, \boldsymbol{w}_t - \boldsymbol{w}^* \rangle \tag{A.16}$$

$$= \eta^2 \|A_t \boldsymbol{\alpha}_t\|^2 - 2\eta \langle A_t \boldsymbol{\alpha}_t, \boldsymbol{w}_t - \hat{\boldsymbol{w}}_t^n \rangle - 2\eta \langle A_t \boldsymbol{\alpha}_t, \hat{\boldsymbol{w}}_t^n - \boldsymbol{w}^* \rangle, \tag{A.17}$$

$$= \eta^2 \|A_t \boldsymbol{\alpha}_t\|^2 - 2\eta \langle A_t \boldsymbol{\alpha}_t, \boldsymbol{w}_t - \hat{\boldsymbol{w}}_t^n \rangle - 2\eta \sum_{n=1}^{N-1} \alpha^n \nabla \ell_{z_t}(\hat{\boldsymbol{w}}_t^n)^\top (\hat{\boldsymbol{w}}_t^n - \boldsymbol{w}^*), (A_t \alpha_t^N = 0) \tag{A.18}$$

$$\leq \eta^2 \|A_t \boldsymbol{\alpha}_t\|^2 - 2\eta \langle A_t \boldsymbol{\alpha}_t, \boldsymbol{w}_t - \hat{\boldsymbol{w}}_t^n \rangle - 2\eta \sum_{n=1}^{N-1} \alpha_t^n (\ell_{z_t}(\hat{\boldsymbol{w}}_t^n) - \ell_{z_t}(\boldsymbol{w}^*)), \quad \text{(convexity)} \tag{A.19}$$

$$\leq \eta^2 \|A_t \boldsymbol{\alpha}_t\|^2 - 2\eta \sum_{n=1}^{N-1} \alpha^n \nabla \ell_{z_t}(\hat{\boldsymbol{w}}_t^n)^\top (\boldsymbol{w}_t - \hat{\boldsymbol{w}}_t^n) - 2\eta \sum_{n=1}^{N-1} \alpha_t^n (\ell_{z_t}(\hat{\boldsymbol{w}}_t^n) - \ell_{z_t}(\boldsymbol{w}^*)), \tag{A.20}$$

$$\leq \eta^2 \|A_t \boldsymbol{\alpha}_t\|^2 - 2\eta \sum_{n=1}^{N-1} \alpha^n [\ell_{z_t}(\hat{\boldsymbol{w}}_t^n) - \nabla \ell_{z_t}(\hat{\boldsymbol{w}}_t^n)^\top (\hat{\boldsymbol{w}}_t^n - \boldsymbol{w}_t)] + 2\eta \sum_{n=1}^{N-1} \alpha_t^n \ell_{z_t}(\boldsymbol{w}^*), \tag{A.21}$$

$$\leq \eta^2 \|A_t \boldsymbol{\alpha}_t\|^2 - 2\eta \boldsymbol{\alpha}_t \boldsymbol{b}_t - 2\eta \sum_{n=1}^{N-1} \alpha_t^n \ell_{z_t}(\boldsymbol{w}^*), \qquad (\boldsymbol{b}_t \text{ definition}) \tag{A.22}$$

$$\leq -2\eta D(\boldsymbol{\alpha}_t) + 2\eta \sum_{n=1}^{N-1} \alpha_t^n \ell_{z_t}(\boldsymbol{w}^*), \qquad (D \text{ definition}) \tag{A.23}$$

$$\leq -2\eta D(\boldsymbol{\alpha}_t) + 2\eta (1 - \alpha_t^N) \ell_{z_t}(\boldsymbol{w}^*), \qquad (\sum_{n=1}^{N} \alpha_t^n = 1) \tag{A.24}$$

$$\leq -2\eta D(\boldsymbol{\alpha}_t), \qquad (\ell_{z_t}(\boldsymbol{w}^*) = 0, \text{ perfect interpolation}) \tag{A.25}$$

$$\leq -2\eta \max_{\boldsymbol{\alpha} \in \Delta_N} D(\boldsymbol{\alpha}) \qquad (\boldsymbol{\alpha}_t \text{definition}) \tag{A.26}$$

□

A consequence of Lemma 2 is that the convergence rate given by Theorem 1 improves as N increases.

A.4.1 Convex and C-Lipschitz

Theorem 1 (Convex and Lipschitz)**.** *Let Ω be a convex set. We assume that for every $z \in \mathcal{Z}$, ℓ_z is convex and C-Lipschitz. Let $\boldsymbol{w}_\star$ be a solution of ($\mathcal{P}$), and assume that we have perfect interpolation: $\forall z \in \mathcal{Z}$, $\ell_z(\boldsymbol{w}_\star) = 0$. Then BORAT for $N \geq 2$ applied to f satisfies:*

$$f\left(\tfrac{1}{T+1}\sum_{t=0}^{T}\boldsymbol{w}_t\right) - f_\star \leq C\sqrt{\frac{\|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2}{(T+1)}} + \frac{\|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2}{\eta(T+1)}. \tag{4.15}$$

Proof. We start from (A.26), hence we have:

$$\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 - 2\eta D(\boldsymbol{\alpha}_t) \tag{A.27}$$

From Lemma 2 we additionally have that $D^2(\boldsymbol{\alpha}_*) \leq D^{N>2}(\boldsymbol{\alpha}_*)$ hence, we consider $N = 2$ as this provides the worst rate.

$$\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 - 2\eta D^2(\boldsymbol{\alpha}_t) \tag{A.28}$$

For $N = 2$ we have exactly two subproblems, and hence we can write the dual in the following compact form:

$$D^2(\boldsymbol{\alpha}_t) = \begin{cases} -\frac{\eta}{2}\|\boldsymbol{g}_{z_t}\|^2 + \ell_{z_t}(\boldsymbol{w}_t), & \text{if } \eta\|\boldsymbol{g}_{z_t}\|^2 \leq \ell_{z_t}(\boldsymbol{w}_t) \\ \frac{1}{2\eta}\frac{\ell_{z_t}(\boldsymbol{w}_t)^2}{\|\boldsymbol{g}_{z_t}\|^2} & \text{if } \eta\|\boldsymbol{g}_{z_t}\|^2 \geq \ell_{z_t}(\boldsymbol{w}_t) \end{cases} \tag{A.29}$$

We now introduce $\mathcal{I}_T$ and $\mathcal{J}_T$ as follows:

$$\mathcal{I}_T \triangleq \left\{t \in \{0, ..., T\} : \eta\|\boldsymbol{g}_{z_t}\|^2 \geq \ell_{z_t}(\boldsymbol{w}_t)\right\}$$
$$\mathcal{J}_T \triangleq \{0, ..., T\} \setminus \mathcal{I}_T$$

Defining $\mathbb{1}$ to be the indicator function we can write:

$$\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 + \mathbb{1}(t \in \mathcal{I}_T)\eta\left(\eta\|\boldsymbol{g}_{z_t}\|^2 - 2\ell_{z_t}(\boldsymbol{w}_t)\right) - \mathbb{1}(t \in \mathcal{J}_T)\left(\frac{\ell_{z_t}(\boldsymbol{w}_t)^2}{\|\boldsymbol{g}_{z_t}\|^2}\right). \tag{A.30}$$

From our definition of $\mathcal{I}_T$ for all $t \in \mathcal{I}_T$ we have $\eta\|\boldsymbol{g}_{z_t}\|^2 \geq \ell_{z_t}(\boldsymbol{w}_t)$, hence, we can write:

$$\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 - \mathbb{1}(t \in \mathcal{I}_T)\eta\ell_{z_t}(\boldsymbol{w}_t) - \mathbb{1}(t \in \mathcal{J}_T)\left(\frac{\ell_{z_t}(\boldsymbol{w}_t)^2}{\|\boldsymbol{g}_{z_t}\|^2}\right). \tag{A.31}$$

Summing over T:

$$\|\boldsymbol{w}_{T+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2 - \eta\sum_{t\in\mathcal{I}_T}\ell_{z_t}(\boldsymbol{w}_t) - \sum_{t\in\mathcal{J}_T}\left(\frac{\ell_{z_t}(\boldsymbol{w}_t)^2}{\|\boldsymbol{g}_{z_t}\|^2}\right) \tag{A.32}$$

Using $\|\boldsymbol{w}_{T+1} - \boldsymbol{w}_\star\|^2 \geq 0$, we obtain:

$$\eta\sum_{t\in\mathcal{I}_T}\ell_{z_t}(\boldsymbol{w}_t) + \sum_{t\in\mathcal{J}_T}\left(\frac{\ell_{z_t}(\boldsymbol{w}_t)^2}{\|\boldsymbol{g}_{z_t}\|^2}\right) \leq \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2. \tag{A.33}$$

From $\left(\frac{\ell_{z_t}(\boldsymbol{w}_t)^2}{\|\boldsymbol{g}_{z_t}\|^2}\right) \geq 0$, we get:

$$\sum_{t\in\mathcal{I}_T}\ell_{z_t}(\boldsymbol{w}_t) \leq \frac{1}{\eta}\|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2. \tag{A.34}$$

Likewise, using the observation that $\ell_z \geq 0$, we can write:

$$\sum_{t\in\mathcal{J}_T}\frac{\ell_{z_t}(\boldsymbol{w}_t)^2}{C^2} \leq \sum_{t\in\mathcal{J}_T}\frac{\ell_{z_t}(\boldsymbol{w}_t)^2}{\|\boldsymbol{g}_{z_t}\|^2} \leq \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2. \tag{A.35}$$

Dividing by C^2:

$$\sum_{t \in \mathcal{J}_T} \ell_{z_t}(\boldsymbol{w}_t)^2 \leq C^2 \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2. \tag{A.36}$$

Using the Cauchy-Schwarz inequality, we can further write:

$$\left(\sum_{t \in \mathcal{J}_T} \ell_{z_t}(\boldsymbol{w}_t) \right)^2 \leq |\mathcal{J}_T| \sum_{t \in \mathcal{J}_T} \ell_{z_t}(\boldsymbol{w}_t)^2. \tag{A.37}$$

Therefore, we have:

$$\sum_{t \in \mathcal{J}_T} \ell_{z_t}(\boldsymbol{w}_t) \leq C \sqrt{|\mathcal{J}_T| \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2}. \tag{A.38}$$

We can now put together inequalities (A.34) and (A.38) by writing:

$$\sum_{t=0}^{T} \ell_{z_t}(\boldsymbol{w}_t) = \sum_{t \in \mathcal{J}_T} \ell_{z_t}(\boldsymbol{w}_t) + \sum_{t \in \mathcal{I}_T} \ell_{z_t}(\boldsymbol{w}_t) \tag{A.39}$$

$$\sum_{t=0}^{T} \ell_{z_t}(\boldsymbol{w}_t) \leq \frac{1}{\eta} \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2 + C \sqrt{|\mathcal{J}_T| \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2} \tag{A.40}$$

$$\sum_{t=0}^{T} \ell_{z_t}(\boldsymbol{w}_t) \leq \frac{1}{\eta} \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2 + C \sqrt{(T+1) \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2} \tag{A.41}$$

Dividing by $T+1$ and taking the expectation over z_t, we finally get:

$$f\left(\frac{1}{T+1} \sum_{t=0}^{T} \boldsymbol{w}_t \right) - f_\star \leq \frac{1}{T+1} \sum_{t=0}^{T} f(\boldsymbol{w}_t) - f_\star, \quad (f \text{ is convex}) \tag{A.42}$$

$$f\left(\frac{1}{T+1} \sum_{t=0}^{T} \boldsymbol{w}_t \right) - f_\star \leq C \sqrt{\frac{\|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2}{(T+1)}} + \frac{\|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2}{\eta(T+1)}. \tag{A.43}$$

□

A.4.2 Convex and Smooth

Lemma 3. *Let $z \in \mathcal{Z}$. Assume that ℓ_z is β-smooth and non-negative on $\mathbb{R}^d$. Then we have:*

$$\forall (\boldsymbol{w}) \in \mathbb{R}^d, \;\; \ell_z(\boldsymbol{w}) \geq \frac{1}{2\beta} \|\nabla \ell_z(\boldsymbol{w})\|^2$$

Note that we do not assume that ℓ_z is convex.

Proof. Let $\boldsymbol{w} \in \mathbb{R}^d$. By Lemma 3.4 of Bubeck (2015), we have:

$$\forall\, \boldsymbol{u} \in \mathbb{R}^d, \;\; |\ell_z(\boldsymbol{u}) - \ell_z(\boldsymbol{w}) - \nabla \ell_z(\boldsymbol{w})^\intercal (\boldsymbol{u} - \boldsymbol{w})| \leq \frac{\beta}{2} \|\boldsymbol{u} - \boldsymbol{w}\|^2.$$

Therefore, we can write:

$$\forall\, \boldsymbol{u} \in \mathbb{R}^d, \;\; \ell_z(\boldsymbol{u}) \leq \ell_z(\boldsymbol{w}) + \nabla \ell_z(\boldsymbol{w})^\intercal (\boldsymbol{u} - \boldsymbol{w})| + \frac{\beta}{2} \|\boldsymbol{u} - \boldsymbol{w}\|^2.$$

Since $\forall\, \boldsymbol{u}, l_z(\boldsymbol{u}) \geq 0$, we have:

$$\forall\, \boldsymbol{u} \in \mathbb{R}^d, \;\; 0 \leq \ell_z(\boldsymbol{w}) + \nabla \ell_z(\boldsymbol{w})^\intercal (\boldsymbol{u} - \boldsymbol{w})| + \frac{\beta}{2} \|\boldsymbol{u} - \boldsymbol{w}\|^2.$$

We now choose $\boldsymbol{u} = -\frac{1}{\beta} \nabla l_z(\boldsymbol{w})$, which yields:

$$\forall\, \boldsymbol{u} \in \mathbb{R}^d, \;\; 0 \leq \ell_z(\boldsymbol{w}) - \frac{1}{\beta} \|\nabla \ell_z(\boldsymbol{w})\|^2 + \frac{\beta}{2} \|\nabla \ell_z(\boldsymbol{w})\|^2,$$

which gives the desired result. □

Theorem 3 (Convex and Smooth - Large η). *We assume that Ω is a convex set, and that for every $z \in \mathcal{Z}$, ℓ_z is convex and β-smooth. Let $\boldsymbol{w}_\star$ be a solution of ($\mathcal{P}$), and assume that we have perfect interpolation: $\forall z \in \mathcal{Z}$, $\ell_z(\boldsymbol{w}_\star) = 0$. Then BORAT*

for $N \geq 2$ applied to f with $\eta \geq \frac{1}{2\beta}$ satisfies:

$$f\left(\tfrac{1}{T+1}\sum_{t=0}^{T}\boldsymbol{w}_t\right) - f_\star \leq 2\beta\frac{\|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2}{T+1}. \tag{A.44}$$

Proof. We start from Equation (A.32) we have:

$$\|\boldsymbol{w}_{T+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2 - \eta\sum_{t\in\mathcal{I}_T}\ell_{z_t}(\boldsymbol{w}_t) - \sum_{t\in\mathcal{J}_T}\left(\frac{\ell_{z_t}(\boldsymbol{w}_t)^2}{\|\boldsymbol{g}_{z_t}\|^2}\right). \tag{A.45}$$

Now using Lemma 3 we can write:

$$\|\boldsymbol{w}_{T+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2 - \eta\sum_{t\in\mathcal{I}_T}\ell_{z_t}(\boldsymbol{w}_t) - \frac{1}{2\beta}\sum_{t\in\mathcal{J}_T}\ell_{z_t}(\boldsymbol{w}_t). \tag{A.46}$$

From our assumption on $\eta \geq \frac{1}{2\beta}$ we can write:

$$\|\boldsymbol{w}_{T+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2 - \frac{1}{2\beta}\sum_{t\in\mathcal{I}_T}\ell_{z_t}(\boldsymbol{w}_t) - \frac{1}{2\beta}\sum_{t\in\mathcal{J}_T}\ell_{z_t}(\boldsymbol{w}_t), \tag{A.47}$$

$$\|\boldsymbol{w}_{T+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2 - \frac{1}{2\beta}\sum_{t=0}^{T}\ell_{z_t}(\boldsymbol{w}_t). \tag{A.48}$$

Using $\|\boldsymbol{w}_{T+1} - \boldsymbol{w}_\star\|^2 \geq 0$, we obtain:

$$\sum_{t=0}^{T}\ell_{z_t}(\boldsymbol{w}_t) \leq 2\beta\|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2. \tag{A.49}$$

Dividing by $T+1$ and taking the expectation over z_t, we finally get:

$$f\left(\frac{1}{T+1}\sum_{t=0}^{T}\boldsymbol{w}_t\right) - f_\star \leq \frac{1}{T+1}\sum_{t=0}^{T}f(\boldsymbol{w}_t) - f_\star, \quad (f \text{ is convex})$$
$$f\left(\frac{1}{T+1}\sum_{t=0}^{T}\boldsymbol{w}_t\right) - f_\star \leq 2\beta\frac{\|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2}{T+1}.$$

□

Theorem 4 (Convex and Smooth - Small η). *We assume that Ω is a convex set, and that for every $z \in \mathcal{Z}$, ℓ_z is convex and β-smooth. Let $\boldsymbol{w}_\star$ be a solution of ($\mathcal{P}$), and assume that we have perfect interpolation: $\forall z \in \mathcal{Z}$, $\ell_z(\boldsymbol{w}_\star) = 0$. Then BORAT for $N \geq 2$ applied to f with $\eta \leq \frac{1}{2\beta}$ satisfies:*

$$f\left(\tfrac{1}{T+1}\sum_{t=0}^{T}\boldsymbol{w}_t\right) - f_\star \leq 2\beta\frac{\|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2}{T+1}. \tag{A.50}$$

Proof. Now considering the $\eta \leq \frac{1}{2\beta}$ starting from (A.51)

$$\|\boldsymbol{w}_{T+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2 - \eta\sum_{t\in\mathcal{I}_T}\ell_{z_t}(\boldsymbol{w}_t) - \frac{1}{2\beta}\sum_{t\in\mathcal{J}_T}\ell_{z_t}(\boldsymbol{w}_t), \tag{A.51}$$

$$\|\boldsymbol{w}_{T+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2 - \eta\sum_{t\in\mathcal{I}_T}\ell_{z_t}(\boldsymbol{w}_t) - \eta\sum_{t\in\mathcal{J}_T}\ell_{z_t}(\boldsymbol{w}_t), \tag{A.52}$$

$$\|\boldsymbol{w}_{T+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2 - \eta\sum_{t=0}^{T}\ell_{z_t}(\boldsymbol{w}_t). \tag{A.53}$$

Using $\|\boldsymbol{w}_{T+1} - \boldsymbol{w}_\star\|^2 \geq 0$, we obtain:

$$\sum_{t=0}^{T}\ell_{z_t}(\boldsymbol{w}_t) \leq \frac{1}{\eta}\|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2. \tag{A.54}$$

Dividing by $T+1$ and taking the expectation over z_t, we finally get:

$$f\left(\frac{1}{T+1}\sum_{t=0}^{T}\boldsymbol{w}_t\right) - f_\star \leq \frac{1}{T+1}\sum_{t=0}^{T}f(\boldsymbol{w}_t) - f_\star, \quad (f \text{ is convex})$$
$$f\left(\frac{1}{T+1}\sum_{t=0}^{T}\boldsymbol{w}_t\right) - f_\star \leq \frac{\|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2}{\eta(T+1)}.$$

□

A.4.3 Strongly Convex

Finally, we consider the α-strongly convex and β-smooth case.

Lemma 4. *Let $z \in \mathcal{Z}$. Assume that ℓ_z is α-strongly convex, non-negative on $\mathbb{R}^d$, and such that $\inf \ell_z = 0$. Then we have:*

$$\forall\, \boldsymbol{w} \in \mathbb{R}^d,\ \frac{\ell_z(\boldsymbol{w})}{\|\nabla \ell_z(\boldsymbol{w})\|^2} \leq \frac{1}{2\alpha}. \tag{A.55}$$

Proof. Let $\boldsymbol{w} \in \mathbb{R}^d$ and suppose that ℓ_z reaches its minimum at $\underline{\mathbf{w}} \in \mathbb{R}^d$ (this minimum exists because of strong convexity). By definition of strong convexity, we have that:

$$\forall\, \hat{\boldsymbol{w}} \in \mathbb{R}^d,\ \ell_z(\hat{\boldsymbol{w}}) \geq \ell_z(\boldsymbol{w}) + \nabla \ell_z(\boldsymbol{w})^\top (\hat{\boldsymbol{w}} - \boldsymbol{w}) + \frac{\alpha}{2}\|\hat{\boldsymbol{w}} - \boldsymbol{w}\|^2 \tag{A.56}$$

We minimize the right hand side over $\hat{\boldsymbol{w}}$, which gives:

$$\begin{aligned} \forall \hat{\boldsymbol{w}} \in \mathbb{R}^d,\ \ell_z(\hat{\boldsymbol{w}}) &\geq \ell_z(\boldsymbol{w}) + \nabla \ell_z(\boldsymbol{w})^\top (\hat{\boldsymbol{w}} - \boldsymbol{w}) + \frac{\alpha}{2}\|\hat{\boldsymbol{w}} - \boldsymbol{w}\|^2 \\ &\geq \ell_z(\boldsymbol{w}) - \frac{1}{2\alpha}\|\nabla \ell_z(\boldsymbol{w})\|^2 \end{aligned} \tag{A.57}$$

Thus, by choosing $\hat{\boldsymbol{w}} = \underline{\mathbf{w}}$ and re-ordering, we obtain the following result (a.k.a. the Polyak-Lojasiewicz inequality):

$$\ell_z(\boldsymbol{w}) - \ell_z(\underline{\mathbf{w}}) \leq \frac{1}{2\alpha}\|\nabla \ell_z(\boldsymbol{w})\|^2 \tag{A.58}$$

However, we have $\ell_z(\underline{\mathbf{w}}) = 0$ which concludes the proof. □

Lemma 5. *For any $a, b \in \mathbb{R}^d$, we have that:*

$$\|a\|^2 + \|b\|^2 \geq \frac{1}{2}\|a - b\|^2. \tag{A.59}$$

Proof. This is a simple application of the parallelogram law, but we give the proof

here for completeness.

$$\begin{aligned}\|a\|^2 + \|b\|^2 - \frac{1}{2}\|a-b\|^2 &= \|a\|^2 + \|b\|^2 - \frac{1}{2}\|a\|^2 - \frac{1}{2}\|b\|^2 + a^\top b \\ &= \frac{1}{2}\|a\|^2 + \frac{1}{2}\|b\|^2 + a^\top b \\ &= \frac{1}{2}\|a+b\|^2 \\ &\geq 0\end{aligned}$$

□

Lemma 6. *Let $z \in \mathcal{Z}$. Assume that ℓ_z is α-strongly convex and achieves its (possibly constrained) minimum at $\boldsymbol{w}_\star \in \Omega$. Then we have:*

$$\forall\, \boldsymbol{w} \in \Omega,\ \ell_z(\boldsymbol{w}) - \ell_z(\boldsymbol{w}_\star) \geq \frac{\alpha}{2}\|\boldsymbol{w} - \boldsymbol{w}_\star\|^2 \tag{A.60}$$

Proof. By definition of strong-convexity (Bubeck, 2015), we have:

$$\forall\, \boldsymbol{w} \in \Omega,\ \ell_z(\boldsymbol{w}) - \ell_z(\boldsymbol{w}_\star) - \nabla\ell_z(\boldsymbol{w}_\star)^\top(\boldsymbol{w} - \boldsymbol{w}_\star) \geq \frac{\alpha}{2}\|\boldsymbol{w} - \boldsymbol{w}_\star\|^2. \tag{A.61}$$

In addition, since $\boldsymbol{w}_\star$ minimises ℓ_z, then necessarily:

$$\forall\, \boldsymbol{w} \in \Omega,\ \nabla\ell_z(\boldsymbol{w}_\star)^\top(\boldsymbol{w} - \boldsymbol{w}_\star) \geq 0. \tag{A.62}$$

Combining the two equations gives the desired result. □

Finally, we consider the α-strongly convex and β-smooth case. Again, our proof yields a natural separation between $\eta \geq \frac{1}{2\beta}$ and $\eta \leq \frac{1}{2\beta}$.

Theorem 5 (Strongly Convex - Large with large η). *We assume that Ω is a convex set, and that for every $z \in \mathcal{Z}$, ℓ_z is α-strongly convex and β-smooth. Let $\boldsymbol{w}_\star$ be a solution of ($\mathcal{P}$), and assume that we have perfect interpolation: $\forall z \in \mathcal{Z}$, $\ell_z(\boldsymbol{w}_\star) = 0$.*

Then BORAT for $N \geq 2$ and applied to f with a $\eta \geq \frac{1}{2\beta}$ satisfies:

$$\mathbb{E}[f(\boldsymbol{w}_{t+1})] - f(\boldsymbol{w}_\star) \leq \frac{\beta}{2} \exp\left(-\frac{\alpha t}{4\beta}\right) \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2. \tag{A.63}$$

Proof. We start from (A.31):

$$\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 - \mathbb{1}(t \in \mathcal{I}_T)\eta\ell_{z_t}(\boldsymbol{w}_t) - \mathbb{1}(t \in \mathcal{J}_T)\left(\frac{\ell_{z_t}(\boldsymbol{w}_t)^2}{\|\boldsymbol{g}_{z_t}\|^2}\right). \tag{A.64}$$

We next use Lemma 3 write:

$$\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 - \mathbb{1}(t \in \mathcal{I}_T)\eta\ell_{z_t}(\boldsymbol{w}_t) - \mathbb{1}(t \in \mathcal{J}_T)\frac{1}{2\beta}\ell_{z_t}(\boldsymbol{w}_t). \tag{A.65}$$

Now we use our assumption on η to give:

$$\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 - \frac{1}{2\beta}\ell_{z_t}(\boldsymbol{w}_t). \tag{A.66}$$

Taking expectations:

$$\mathbb{E}[\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2] \leq \|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 - \frac{1}{2\beta}f(\boldsymbol{w}_t). \tag{A.67}$$

Using Lemma 6:

$$\mathbb{E}[\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2] \leq \|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 - \frac{\alpha}{4\beta}\|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2. \tag{A.68}$$

We use a trivial induction over t and write:

$$\mathbb{E}[\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2] \leq \left(1 - \frac{\alpha}{4\beta}\right) \|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2, \tag{A.69}$$

$$\mathbb{E}[\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2] \leq \left(1 - \frac{\alpha}{4\beta}\right)^t \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2. \tag{A.70}$$

$$\tag{A.71}$$

Given an arbitrary $\boldsymbol{w} \in \mathbb{R}^d$, we now wish to relate the distance $\|\boldsymbol{w} - \boldsymbol{w}_\star\|^2$ to the function values $f(\boldsymbol{w}) - f(\boldsymbol{w}_\star)$. From smoothness, we have:

$$f(\boldsymbol{w}_{t+1}) - f(\boldsymbol{w}_\star) \leq \nabla f(\boldsymbol{w}_\star)^\top (\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star) + \frac{\beta}{2}\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2. \tag{A.72}$$

However, we know by definition $\nabla f(\boldsymbol{w}_\star) = 0$, hence:

$$f(\boldsymbol{w}_{t+1}) - f(\boldsymbol{w}_\star) \leq \frac{\beta}{2}\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2. \tag{A.73}$$

Taking expectations:

$$\mathbb{E}[f(\boldsymbol{w}_{t+1})] - f(\boldsymbol{w}_\star) \leq \frac{\beta}{2}\mathbb{E}[\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2]. \tag{A.74}$$

Hence, we recover the final rate:

$$\mathbb{E}[f(\boldsymbol{w}_{t+1})] - f(\boldsymbol{w}_\star) \leq \frac{\beta}{2}\left(1 - \frac{\alpha}{4\beta}\right)^t \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2, \tag{A.75}$$

$$\mathbb{E}[f(\boldsymbol{w}_{t+1})] - f(\boldsymbol{w}_\star) \leq \frac{\beta}{2}\exp\left(-\frac{\alpha t}{4\beta}\right) \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2. \tag{A.76}$$

□

Theorem 6 (Strongly Convex - Small η). *We assume that Ω is a convex set, and that for every $z \in \mathcal{Z}$, ℓ_z is α-strongly convex and β-smooth. Let $\boldsymbol{w}_\star$ be a solution of ($\mathcal{P}$), and assume that we have perfect interpolation: $\forall z \in \mathcal{Z}$, $\ell_z(\boldsymbol{w}_\star) = 0$. Then*

BORAT for $N \geq 2$ and applied to f with a $\eta \leq \frac{1}{2\beta}$ satisfies:

$$\mathbb{E}[f(\boldsymbol{w}_{t+1})] - f(\boldsymbol{w}_\star) \leq \frac{\beta}{2} \exp\left(-\frac{\alpha\eta t}{2}\right) \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2. \tag{A.77}$$

Proof. We start from (A.31):

$$\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 - \mathbb{1}(t \in \mathcal{I}_T)\eta\ell_{z_t}(\boldsymbol{w}_t) - \mathbb{1}(t \in \mathcal{J}_T)\left(\frac{\ell_{z_t}(\boldsymbol{w}_t)^2}{\|\boldsymbol{g}_{z_t}\|^2}\right). \tag{A.78}$$

We next use Lemma 3 write:

$$\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 - \mathbb{1}(t \in \mathcal{I}_T)\eta\ell_{z_t}(\boldsymbol{w}_t) - \mathbb{1}(t \in \mathcal{J}_T)\frac{1}{2\beta}\ell_{z_t}(\boldsymbol{w}_t). \tag{A.79}$$

Now we use our assumption on η to give:

$$\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 - \eta\ell_{z_t}(\boldsymbol{w}_t) \tag{A.80}$$

Taking expectations:

$$\mathbb{E}[\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2] \leq \|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 - \eta\ell_{z_t}(\boldsymbol{w}_t) \tag{A.81}$$

Using Lemma 6

$$\mathbb{E}[\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2] \leq \|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 - \frac{\alpha\eta}{2}\|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 \tag{A.82}$$

We use a trivial induction over t and write:

$$\mathbb{E}[\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2] \leq \left(1 - \frac{\alpha\eta}{2}\right)\|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2, \tag{A.83}$$

$$\mathbb{E}[\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2] \leq \left(1 - \frac{\alpha\eta}{2}\right)^t \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2, \tag{A.84}$$

$$\tag{A.85}$$

Given an arbitrary $\boldsymbol{w} \in \mathbb{R}^d$, we now wish to relate the distance $\|\boldsymbol{w} - \boldsymbol{w}_\star\|^2$ to the function values $f(\boldsymbol{w}) - f(\boldsymbol{w}_\star)$. From smoothness, we have:

$$f(\boldsymbol{w}_{t+1}) - f(\boldsymbol{w}_\star) \leq \nabla f(\boldsymbol{w}_\star)^\top (\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star) + \frac{\beta}{2}\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2. \tag{A.86}$$

However, we know by definition $\nabla f(\boldsymbol{w}_\star) = 0$, hence:

$$f(\boldsymbol{w}_{t+1}) - f(\boldsymbol{w}_\star) \leq \frac{\beta}{2}\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2. \tag{A.87}$$

Taking expectations:

$$\mathbb{E}[f(\boldsymbol{w}_{t+1})] - f(\boldsymbol{w}_\star) \leq \frac{\beta}{2}\mathbb{E}[\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2]. \tag{A.88}$$

Hence, we recover the final rate:

$$\mathbb{E}[f(\boldsymbol{w}_{t+1})] - f(\boldsymbol{w}_\star) \leq \frac{\beta}{2}\left(1 - \frac{\alpha\eta}{2}\right)^t \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2, \tag{A.89}$$

$$\mathbb{E}[f(\boldsymbol{w}_{t+1})] - f(\boldsymbol{w}_\star) \leq \frac{\beta}{2}\exp\left(\frac{-\alpha\eta t}{2}\right) \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2. \tag{A.90}$$

□

A.5 Non-Convex Results

Here we provide the proof of Theorem 2, which we restate for clarity. To simplify our analysis, we consider the BORAT algorithm with $N = 3$. We prove these results for BORAT with the minor modification that all linear approximations are formed using the same mini-batch of data, $\ell_{z_t^n} = \ell_{z_t}$ for all $n \in \{2, ..., N-1\}$.

Theorem 2 (RSI)**.** *We consider problems of type (P). We assume ℓ_z satisfies the RSI with constant μ, smoothness constant β and perfect interpolation e.g. $l_z(\boldsymbol{w}^*) =$*

0, $\forall z \in \mathcal{Z}$. *Then if set* $\eta \leq \hat{\eta} = \min\{\frac{1}{4\beta}, \frac{1}{4\mu}, \frac{\mu}{2\beta^2}\}$ *then in the worst case we have:*

$$f(\boldsymbol{w}_{T+1}) - f^* \leq exp\left(\left(-\frac{3}{8}\hat{\eta}\mu\right)T\right)||\boldsymbol{w}_0 - \boldsymbol{w}^*||^2. \tag{4.17}$$

In order to derive the proof for Theorem 2 we first give a brief overview of BORAT with $N = 3$. We detail the $(2^N - 1)$ possible subproblems (7 in this case), and the resulting values of $\boldsymbol{\alpha}_t$ for each. We show for $\eta \leq \frac{1}{2\beta}$, only a sub-set of the subproblems result in valid solutions with optimal points within the simplex. Finally, we derive a rate that we assume is optimal for all t for each of the remaining subproblems Lastly, by taking the minimum of these rates we construct the worst case rate.

A.5.1 BORAT with $N = 3$

With $N = 3$ the bundle is made of three linear approximations. These are the lower bound on the loss, a linear approximations made at the current point, and at the optimum of the bundle of size two. Hence, this third linear approximation is made at the location one would reach after taking an ALI-G step. Note, here we use γ_t to denote the optimal value of α^1 as given by 3.15. As some of these proofs get quite involved, we make use of the following abbreviations to save space:

$$\begin{aligned}
&\hat{\boldsymbol{w}}_t^1 = \boldsymbol{w}_t, \qquad &&\boldsymbol{g}_{z_t} = \nabla\ell_{z_t^n}(\boldsymbol{w}_t), \qquad &&b_t^1 = \ell_{z_t^n}(\boldsymbol{w}_t),\\
&\hat{\boldsymbol{w}}_t^2 = \boldsymbol{w}'_t, \qquad &&\boldsymbol{g}'_{z_t} = \nabla\ell_{z_t^n}(\boldsymbol{w}'_t), \qquad &&b_t^2 = \ell_{z_t^n}(\boldsymbol{w}'_t) + \eta\gamma_t\langle\boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t}\rangle.
\end{aligned}$$

Where $\boldsymbol{w}'_t$ is defined as $\boldsymbol{w}'_t = \boldsymbol{w}_t - \eta\gamma_t\nabla\ell_{z_t^n}(\boldsymbol{w}_t)$, where $\gamma_t = \min\{1, \frac{\ell_{z_t^n}(\boldsymbol{w}_t)}{\eta\|\nabla\ell_{z_t^n}(\boldsymbol{w}_t)\|^2}\}$. With this notation defined, the BORAT primal problem for this special case can be

simplified to:

$$\boldsymbol{w}_{t+1} = \underset{\boldsymbol{w}\in\mathbb{R}^d}{\operatorname{argmin}} \left\{ \frac{1}{2\eta}||\boldsymbol{w} - \boldsymbol{w}_t||^2 + \max\left\{ \langle \boldsymbol{g}_{z_t}, \boldsymbol{w} - \boldsymbol{w}_t\rangle + b_t^1, \langle \boldsymbol{g}'_{z_t}, \boldsymbol{w} - \boldsymbol{w}_t\rangle + b_t^2, 0 \right\} \right\}. \tag{A.91}$$

The dual of (A.91) can be written as:

$$\boldsymbol{\alpha}_t = \underset{\boldsymbol{\alpha}\in\Delta_3}{\operatorname{argmax}} \left\{ -\frac{\eta}{2}||\alpha^1 \boldsymbol{g}_{z_t} + \alpha^{2t}\boldsymbol{g}'_{z_t}||^2 + \alpha^1 \ell_{z_t^n}(\boldsymbol{w}_t) + \alpha^2 \ell_{z_t^n}(\boldsymbol{w}'_t) + \alpha^2 \eta\gamma_t \langle \boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t}\rangle \right\}. \tag{A.92}$$

For $N = 3$ we have the following parameter update:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha^{1t}\eta\nabla\ell_{z_t^n}(\boldsymbol{w}_t) - \alpha^{2t}\eta\nabla\ell_{z_t^n}(\boldsymbol{w}'_t), \tag{A.93}$$

A.5.2 Subproblems

Algorithm 1 solves $(2^N - 1)$ subproblems, Each of these linear systems corresponds to one of the loci of the simplex, defining the feasible set. We refer to each of these $(2^N - 1)$ options as subproblems. However, each subproblem can also be interpreted as a sub-system of $Q\boldsymbol{\alpha} = \boldsymbol{b}$, (see line 2 of Algorithm 1 for definitions). For $N = 3$ the form of $Q\boldsymbol{\alpha} = \boldsymbol{b}$ is detailed in (A.94).

$$\begin{bmatrix} \eta||\boldsymbol{g}_{z_t}||^2 & \eta\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t}\rangle & 0 & 1 \\ \eta\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t}\rangle & \eta||\boldsymbol{g}'_{z_t}||^2 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha^1 \\ \alpha^2 \\ \alpha^3 \\ c \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ 0 \\ 1 \end{bmatrix}, \tag{A.94}$$

where c is defined in Theorem 1. Each subproblem is defined by setting a unique subset of the dual variables α^n to zero, before solving for the remaining variables. For $(N = 3)$ we have exactly seven subproblems, which we each give a unique label, see

Table A.1. For clarity, we detail a few specific subproblems. The SGD subproblem corresponds to setting $\alpha^2, \alpha^3 = 0$, and recovers the SGD update. Likewise, the ESGD step corresponds to setting $\alpha^1, \alpha^3 = 0$ and recovers an update similar to the extra gradient method. Finally, by setting $\alpha^2 = 0$ before solving the system we recover an ALI-G like step, hence, we label this subproblem as the ALI-G step. If a subproblem results in a $\boldsymbol{\alpha} \in \Delta_3$ we refer to that subproblem as feasible, conversely, if it results in a $\boldsymbol{\alpha} \notin \Delta_3$ we refer to that subproblem as infeasible. Algorithm 2 computes the dual value for all feasible subproblems and selects the largest. This subproblem's $\boldsymbol{\alpha}$ is then used in (A.93) to update the parameters. The closed form solutions for $\boldsymbol{\alpha}$ for each of the 7 subproblems are listed in Table A.1. We use a subscript to show that $\boldsymbol{\alpha}$ belongs to a certain subproblem. For example $\boldsymbol{\alpha}_{SGD} = [1, 0, 0]$.

Table A.1: *Subproblems for $N = 3$.*

α^1	α^2	α^3	Label
1	0	0	SGD
0	1	0	SEGD
0	0	1	ZERO
0	$\frac{b_2}{\eta\|\|\boldsymbol{g}'_{z_t}\|\|^2}$	$1 - \frac{b_2}{\eta\|\|\boldsymbol{g}'_{z_t}\|\|^2}$	EALIG
$\frac{b_1}{\eta\|\|\boldsymbol{g}_{z_t}\|\|^2}$	0	$1 - \frac{b_1}{\eta\|\|\boldsymbol{g}_{z_t}\|\|^2}$	ALIG
$\frac{\eta\|\|\boldsymbol{g}'_{z_t}\|\|^2 - 2\eta\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t}\rangle + b_1 - b_2}{\eta\|\|\boldsymbol{g}_{z_t} - \boldsymbol{g}'_{z_t}\|\|^2}$	$\frac{\eta\|\|\boldsymbol{g}_{z_t}\|\|^2 + b_2 - b_1}{\eta\|\|\boldsymbol{g}_{z_t} - \boldsymbol{g}'_{z_t}\|\|^2}$	0	MAX2
$\frac{b_1\|\|\boldsymbol{g}'_{z_t}\|\|^2 - b_2 \boldsymbol{g}_{z_t}^\top \boldsymbol{g}'_{z_t}}{\eta\|\|\boldsymbol{g}_{z_t}\|\|^2\|\|\boldsymbol{g}'_{z_t}\|\|^2 - \eta\|\|\boldsymbol{g}_{z_t}\boldsymbol{g}'_{z_t}\|\|^2}$	$\frac{b_2\|\|\boldsymbol{g}_{z_t}\|\|^2 - b_1 \boldsymbol{g}_{z_t}^\top a_2}{\eta\|\|\boldsymbol{g}_{z_t}\|\|^2\|\|\boldsymbol{g}'_{z_t}\|\|^2 - \eta\|\|\boldsymbol{g}_{z_t}\boldsymbol{g}'_{z_t}\|\|^2}$	$1 - \alpha^1 - \alpha^2$	MAX3

A.5.3 Dual Values

The corresponding expressions for the dual values for the seven different subproblems are detailed below:

$$
\begin{aligned}
D_{ZERO}(\boldsymbol{\alpha}) &= 0, \\
D_{SGD}(\boldsymbol{\alpha}) &= -\frac{\eta}{2}||\boldsymbol{g}_{z_t}||^2 + \ell_{z_t^n}(\boldsymbol{w}_t), \\
D_{ESGD}(\boldsymbol{\alpha}) &= -\frac{\eta}{2}||\boldsymbol{g}'_{z_t}||^2 + \ell_{z_t^n}(\boldsymbol{w}'_t) + \eta\gamma_t\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t}\rangle, \\
D_{ALIG}(\boldsymbol{\alpha}) &= \frac{1}{2\eta}\frac{\ell_{z_t^n}(\boldsymbol{w}_t)^2}{||\boldsymbol{g}_{z_t}||^2}, \\
D_{EALIG}(\boldsymbol{\alpha}) &= \frac{1}{2\eta}\frac{\left(\ell_{z_t^n}(\boldsymbol{w}'_t) + \eta\gamma_t\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t}\rangle\right)^2}{||\boldsymbol{g}'_{z_t}||^2}, \\
D_{MAX2}(\boldsymbol{\alpha}) &= \frac{1}{2\eta||\boldsymbol{g}_{z_t} - \boldsymbol{g}'_{z_t}||^2}\Bigg((\ell_{z_t^n}(\boldsymbol{w}'_t) - \ell_{z_t^n}(\boldsymbol{w}_t))^2 + 2\eta\left(\ell_{z_t^n}(\boldsymbol{w}'_t)||\boldsymbol{g}_{z_t}||^2 + \ell_{z_t^n}(\boldsymbol{w}_t)||\boldsymbol{g}'_{z_t}||^2\right) \\
&\quad - 4\eta\ell_{z_t^n}(\boldsymbol{w}_t)\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t}\rangle + 2\eta^2||\boldsymbol{g}_{z_t}||^2\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t}\rangle - \eta^2||\boldsymbol{g}_{z_t}||^2||\boldsymbol{g}'_{z_t}||^2\Bigg), \\
D_{MAX3}(\boldsymbol{\alpha}) &= \frac{1}{2}\frac{\left(\ell_{z_t^n}(\boldsymbol{w}'_t)\boldsymbol{g}_{z_t} + \eta\gamma_t\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t}\rangle\boldsymbol{g}_{z_t} - \ell_{z_t^n}(\boldsymbol{w}_t)\boldsymbol{g}'_{z_t}\right)^2}{\eta||\boldsymbol{g}_{z_t}||^2||\boldsymbol{g}'_{z_t}||^2 - \eta\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t}\rangle^2}.
\end{aligned}
$$

The dual value for each subproblem is simply derived by inserting the closed form expression for $\boldsymbol{\alpha}$ for each subproblem detailed in Table A.1 into (4.4). These dual values observe a tree-like hierarchy,

$$
\begin{aligned}
D_{MAX3} &\geq D_{MAX2}, D_{ALIG}, D_{EALIG}, \\
D_{MAX2} &\geq D_{SGD}, D_{ESGD}, \\
D_{ALIG} &\geq D_{SGD}, D_{ZERO}, \\
D_{EALIG} &\geq D_{ESGD}, D_{ZERO}.
\end{aligned}
$$

This hierarchy is a consequence of the subproblems maximising the dual over progressively larger spaces, $\Delta^{n-1} \subset \Delta^n$.

A.5.4 Feasible Subproblems

To give a worst case rate on the convergence of BORAT with $N = 3$, we first prove for smooth functions and small η, only certain subproblems will be feasible. We start with a useful lemma, before proving this result.

Lemma 7. *Let $z \in \mathcal{Z}$. Assume that ℓ_z is β-smooth. If we define $\boldsymbol{w}' = \boldsymbol{w} - \eta\nabla\ell_z(\boldsymbol{w})$ and $\eta \leq \frac{1}{\beta}$ then we have:*

$$\forall(\boldsymbol{w}) \in \mathbb{R}^d, \;\; \langle\nabla\ell_z(\boldsymbol{w}), \nabla\ell_z(\boldsymbol{w}')\rangle \geq 0.$$

Note that we do not assume that ℓ_z is convex.

Proof.

$$\begin{aligned}
2\langle\nabla\ell_z(\boldsymbol{w}), \nabla\ell_z(\boldsymbol{w}')\rangle &= -||\nabla\ell_z(\boldsymbol{w}) - \nabla\ell_z(\boldsymbol{w}')||^2 + ||\nabla\ell_z(\boldsymbol{w})||^2 + ||\nabla\ell_z(\boldsymbol{w}')||^2 \\
2\langle\nabla\ell_z(\boldsymbol{w}), \nabla\ell_z(\boldsymbol{w}')\rangle &\geq -\beta^2||\boldsymbol{w} - \boldsymbol{w}'||^2 + ||\nabla\ell_z(\boldsymbol{w})||^2 + ||\nabla\ell_z(\boldsymbol{w}')||^2, \quad \text{(smoothness)} \\
2\langle\nabla\ell_z(\boldsymbol{w}), \nabla\ell_z(\boldsymbol{w}')\rangle &\geq -\beta^2\eta^2||\nabla\ell_z(\boldsymbol{w})||^2 + ||\nabla\ell_z(\boldsymbol{w})||^2 + ||\nabla\ell_z(\boldsymbol{w}')||^2, \quad (\boldsymbol{w}' \text{ definition}) \\
\langle\nabla\ell_z(\boldsymbol{w}), \nabla\ell_z(\boldsymbol{w}')\rangle &\geq \frac{1}{2}(1 - \beta^2\eta^2)||\nabla\ell_z(\boldsymbol{w})||^2 + \frac{1}{2}||\nabla\ell_z(\boldsymbol{w}')||^2, \\
\langle\nabla\ell_z(\boldsymbol{w}), \nabla\ell_z(\boldsymbol{w}')\rangle &\geq 0. \quad (\frac{1}{\beta} \geq \eta)
\end{aligned}$$

□

Lemma 8 (Feasible subproblems)**.** *If $\ell_{z_t^n}$ has smoothness constant β and we set $\eta \leq \frac{1}{2\beta}$ for the BORAT algorithm with $N = 3$ detailed in Section A.5.1, the ALI-G, EALIG and MAX3 subproblems will always be infeasible.*

Proof. We start by showing the ALI-G step is infeasible. From Lemma 3 we have:

$$\frac{\ell_z(\boldsymbol{w})}{||\boldsymbol{g}_{z_t}||^2} \geq \frac{1}{2\beta}.$$

For the ALI-G step to be feasible we require $\alpha^3_{ALIG} > 0$ or $1 - \frac{\ell_{z_t^n}(\boldsymbol{w}_t)}{\eta||\boldsymbol{g}_{z_t}||^2} > 0$. Rear-

ranging, then we have:

$$\eta > \frac{\ell_{z_t^n}(\boldsymbol{w}_t)}{||\boldsymbol{g}_{z_t}||^2}.$$

Combining these two inequalities gives:

$$\eta > \frac{\ell_z(\boldsymbol{w})}{||\boldsymbol{g}_{z_t}||^2} \geq \frac{1}{2\beta}.$$

Hence, for any $\frac{1}{2\beta} \geq \eta$, $\eta > \frac{\ell_{z_t^n}(\boldsymbol{w}_t)}{||\boldsymbol{g}_{z_t}||^2}$ cannot hold. We now use a similar argument to show that the EALIG subproblem is infeasible. For EALIG to be feasible we require $\alpha^3_{EALIG} > 0$, plugging in the closed form solution for α^3_{EALIG} gives:

$$\begin{aligned}\eta &> \frac{\ell_{z_t^n}(\boldsymbol{w}'_t) + \eta\gamma_t\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t}\rangle}{||\boldsymbol{g}'_{z_t}||^2} = \frac{\ell_{z_t^n}(\boldsymbol{w}'_t) + \eta\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t}\rangle}{||\boldsymbol{g}'_{z_t}||^2} \\ &= \frac{\ell_{z_t^n}(\boldsymbol{w}'_t)}{||\boldsymbol{g}'_{z_t}||^2} + \frac{\eta\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t}\rangle}{||\boldsymbol{g}'_{z_t}||^2} \geq \frac{\ell_{z_t^n}(\boldsymbol{w}'_t)}{||\boldsymbol{g}'_{z_t}||^2} \geq \frac{1}{2\beta},\end{aligned}$$

where the penultimate inequality makes use of $\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t}\rangle \geq 0$, which is a direct application of Lemma 7. The final inequality is a direct application of Lemma 3. Again, it is clear that the condition $\eta > \frac{\ell_{z_t^n}(\boldsymbol{w}'_t)}{||\boldsymbol{g}'_{z_t}||^2}$ cannot be satisfied for $\eta \leq \frac{1}{2\beta}$.
We show that the $MAX3$ step is never taken for $\eta \leq \frac{1}{2\beta}$. First, we show that the dual value for the $MAX3$ step can be written as:

$$D_{MAX3}(\boldsymbol{\alpha}) = \frac{1}{2}\left(\ell_{z_t^n}(\boldsymbol{w}_t)\alpha^{1t} + \ell_{z_t^n}(\boldsymbol{w}'_t)\alpha^{2t} + \eta\gamma_t\alpha^{2t}\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t}\rangle\right).$$

We start from the dual value stated in Section A.5.3.

$$D_{MAX3}(\boldsymbol{\alpha}) = \frac{1}{2}\frac{\left(\ell_{z_t^n}(\boldsymbol{w}'_t)\boldsymbol{g}_{z_t} + \eta\gamma_t\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t}\rangle\boldsymbol{g}_{z_t} - \ell_{z_t^n}(\boldsymbol{w}_t)\boldsymbol{g}'_{z_t}\right)^2}{\eta||\boldsymbol{g}_{z_t}||^2||\boldsymbol{g}'_{z_t}||^2 - \eta\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t}\rangle^2},$$

expanding,

$$
\begin{aligned}
D_{MAX3}(\boldsymbol{\alpha}) = \frac{1}{2}\ell_{z_t^n}(\boldsymbol{w}_t') \underbrace{\frac{\left(\ell_{z_t^n}(\boldsymbol{w}_t')\boldsymbol{g}_{z_t} + \eta\gamma_t\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}_{z_t}'\rangle \boldsymbol{g}_{z_t} - \ell_{z_t^n}(\boldsymbol{w}_t)\boldsymbol{g}_{z_t}'\right)}{\eta||\boldsymbol{g}_{z_t}||^2||\boldsymbol{g}_{z_t}'||^2 - \eta\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}_{z_t}'\rangle^2}}_{=\alpha^2}\boldsymbol{g}_{z_t} \\
+ \frac{1}{2}\eta\gamma_t\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}_{z_t}'\rangle \underbrace{\frac{\left(\ell_{z_t^n}(\boldsymbol{w}_t')\boldsymbol{g}_{z_t} + \eta\gamma_t\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}_{z_t}'\rangle \boldsymbol{g}_{z_t} - \ell_{z_t^n}(\boldsymbol{w}_t)\boldsymbol{g}_{z_t}'\right)}{\eta||\boldsymbol{g}_{z_t}||^2||\boldsymbol{g}_{z_t}'||^2 - \eta\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}_{z_t}'\rangle^2}}_{=\alpha^2}\boldsymbol{g}_{z_t} \\
- \frac{1}{2}\ell_{z_t^n}(\boldsymbol{w}_t) \underbrace{\frac{\left(\ell_{z_t^n}(\boldsymbol{w}_t')\boldsymbol{g}_{z_t} + \eta\gamma_t\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}_{z_t}'\rangle \boldsymbol{g}_{z_t} - \ell_{z_t^n}(\boldsymbol{w}_t)\boldsymbol{g}_{z_t}'\right)}{\eta||\boldsymbol{g}_{z_t}||^2||\boldsymbol{g}_{z_t}'||^2 - \eta\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}_{z_t}'\rangle^2}}_{=-\alpha^1}\boldsymbol{g}_{z_t}'.
\end{aligned}
$$

Using the definitions of α^1_{MAX3} and α^2_{MAX3} we recover the following expression for the $MAX3$ subproblem's dual function:

$$
D_{MAX3}(\boldsymbol{\alpha}) = \frac{1}{2}\left(\ell_{z_t^n}(\boldsymbol{w}_t)\alpha^{1t} + \ell_{z_t^n}(\boldsymbol{w}_t')\alpha^{2t} + \eta\gamma_t\alpha^{2t}\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}_{z_t}'\rangle\right).
$$

With the dual function in this form it is easy to upper bound the feasible dual value for the $MAX3$ subproblem as $D_{MAX3} \leq \frac{1}{2}\max\left\{\ell_{z_t^n}(\boldsymbol{w}_t), \ell_{z_t^n}(\boldsymbol{w}_t') + \eta\gamma_t\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}_{z_t}'\rangle\right\}$. This is a consequence of the fact that $\boldsymbol{\alpha} \in \Delta$ must hold for feasible steps. However, from the hierarchy of dual values we have the lower bounds $D_{MAX3} \geq D_{SGD}$ and $D_{MAX3} \geq D_{ESGD}$, on the $MAX3$ dual value (see Section A.5.3). If either of these lower bounds have a larger value than the feasible dual value's upper bound, the $MAX3$ step will not be feasible. We now show that this is always the case for $\eta \leq \frac{1}{2\beta}$. In order to do this we consider two cases.

We first assume $\ell_{z_t^n}(\boldsymbol{w}_t) \geq \ell_{z_t^n}(\boldsymbol{w}_t') + \eta\gamma_t\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}_{z_t}'\rangle$. Hence, we know the maximum feasible value for $D^{MAX3} = \frac{1}{2}\ell_{z_t^n}(\boldsymbol{w}_t)$, if either D^{SGD} or D^{ESGD} have a larger dual value we can conclude that the $MAX3$ step is infeasible.

$$
D_{SGD}(\boldsymbol{\alpha}) = -\frac{\eta}{2}||\boldsymbol{g}_{z_t}||^2 + \ell_{z_t^n}(\boldsymbol{w}_t),
$$

Hence, if the following condition holds, we know the $MAX3$ step will be infeasible:

$$\frac{1}{2}\ell_{z_t^n}(\boldsymbol{w}_t) \leq -\frac{\eta}{2}||\boldsymbol{g}_{z_t}||^2 + \ell_{z_t^n}(\boldsymbol{w}_t).$$

Thus, the converse must hold for the $MAX3$ step to be feasible:

$$\frac{1}{2}\ell_{z_t^n}(\boldsymbol{w}_t) \geq -\frac{\eta}{2}||\boldsymbol{g}_{z_t}||^2 + \ell_{z_t^n}(\boldsymbol{w}_t),$$

which is equivalent to,

$$\eta \geq \frac{\ell_{z_t^n}(\boldsymbol{w}_t)}{||\boldsymbol{g}_{z_t}||^2}.$$

Using the same logic as stated for the ALI-G step we know this condition is never satisfied for $\eta \leq \frac{1}{2\beta}$.

We now assume $\ell_{z_t^n}(\boldsymbol{w}_t) \leq \ell_{z_t^n}(\boldsymbol{w}'_t) + \eta\gamma_t\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t}\rangle$ and thus we know the max feasible value of $D^{MAX3} \leq \frac{1}{2}\ell_{z_t^n}(\boldsymbol{w}'_t) + \frac{1}{2}\eta\gamma_t\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t}\rangle$, again if either D^{SGD} or D^{ESGD} have larger values, we know the $MAX3$ subproblem is infeasible:

$$D_{ESGD}(\boldsymbol{\alpha}) = -\frac{\eta}{2}||\boldsymbol{g}'_{z_t}||^2 + \ell_{z_t^n}(\boldsymbol{w}'_t) + \eta\gamma_t\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t}\rangle.$$

Hence for the $MAX3$ step to be valid we must have:

$$\frac{1}{2}\ell_{z_t^n}(\boldsymbol{w}'_t) + \eta\gamma_t\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t}\rangle \leq -\frac{\eta}{2}||\boldsymbol{g}'_{z_t}||^2 + \ell_{z_t^n}(\boldsymbol{w}'_t) + \eta\gamma_t\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t}\rangle,$$

which is equivalent to,

$$\eta \leq \frac{\ell_{z_t^n}(\boldsymbol{w}'_t) + \eta\gamma_t \langle \boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t} \rangle}{||\boldsymbol{g}'_{z_t}||^2}.$$

Again, the same condition exist as for the EALIG step, which we have already proven can never be feasible for $\eta \leq \frac{1}{2\beta}$. Hence, the $MAX3$ subproblem is never feasible for $\eta \leq \frac{1}{2\beta}$. □

Lemma 9. *For any set of vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ then, the following inequality holds:*

$$-2||\boldsymbol{a} - \boldsymbol{b}||^2 \leq -||\boldsymbol{a} - \boldsymbol{c}||^2 + 2||\boldsymbol{b} - \boldsymbol{c}||^2.$$

Proof. First consider two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$.

$$\begin{aligned}
0 &\leq ||\boldsymbol{x} - \boldsymbol{y}||^2, \\
0 &\leq ||\boldsymbol{x}||^2 + ||\boldsymbol{y}||^2 - 2\langle \boldsymbol{x}, \boldsymbol{y} \rangle \\
2\langle \boldsymbol{x}, \boldsymbol{y} \rangle &\leq ||\boldsymbol{x}||^2 + ||\boldsymbol{y}||^2, \\
||\boldsymbol{x} + \boldsymbol{y}||^2 &= ||\boldsymbol{x}||^2 + ||\boldsymbol{y}||^2 + 2\langle \boldsymbol{x}, \boldsymbol{y} \rangle, \\
||\boldsymbol{x} + \boldsymbol{y}||^2 &\leq 2||\boldsymbol{x}||^2 + 2||\boldsymbol{y}||^2, \\
-2||\boldsymbol{x}||^2 &\leq 2||\boldsymbol{y}||^2 - ||\boldsymbol{x} + \boldsymbol{y}||^2.
\end{aligned}$$

Setting $\boldsymbol{x} = \boldsymbol{a} - \boldsymbol{b}$ and $\boldsymbol{y} = \boldsymbol{b} - \boldsymbol{c}$ gives the desired result. □

Lemma 10. *Let $z \in \mathcal{Z}$. Assume that ℓ_z is β-smooth and non-negative on $\mathbb{R}^d$. Then we have:*

$$\forall (\boldsymbol{w}) \in \mathbb{R}^d, \;\; \ell_z(\boldsymbol{w}) \geq \frac{1}{2\beta} ||\nabla \ell_z(\boldsymbol{w})||^2$$

Note that we do not assume that ℓ_z is convex.

Proof. Let $\boldsymbol{w} \in \mathbb{R}^d$. By Lemma 3.4 of Bubeck (2015), we have:

$$\forall\, \boldsymbol{u} \in \mathbb{R}^d,\ \ |\ell_z(\boldsymbol{u}) - \ell_z(\boldsymbol{w}) - \nabla\ell_z(\boldsymbol{w})^\intercal(\boldsymbol{u} - \boldsymbol{w})| \leq \frac{\beta}{2}||\boldsymbol{u} - \boldsymbol{w}||^2.$$

Therefore we can write:

$$\forall\, \boldsymbol{u} \in \mathbb{R}^d,\ \ \ell_z(\boldsymbol{u}) \leq \ell_z(\boldsymbol{w}) + \nabla\ell_z(\boldsymbol{w})^\intercal(\boldsymbol{u} - \boldsymbol{w})| + \frac{\beta}{2}||\boldsymbol{u} - \boldsymbol{w}||^2.$$

And since $\forall\, \boldsymbol{u}, l_z(\boldsymbol{u}) \geq 0$, we have:

$$\forall\, \boldsymbol{u} \in \mathbb{R}^d,\ \ 0 \leq \ell_z(\boldsymbol{w}) + \nabla\ell_z(\boldsymbol{w})^\intercal(\boldsymbol{u} - \boldsymbol{w})| + \frac{\beta}{2}||\boldsymbol{u} - \boldsymbol{w}||^2.$$

We now choose $\boldsymbol{u} = -\frac{1}{\beta}\nabla l_z(\boldsymbol{w})$, which yeilds:

$$\forall\, \boldsymbol{u} \in \mathbb{R}^d,\ \ 0 \leq \ell_z(\boldsymbol{w}) - \frac{1}{\beta}||\nabla\ell_z(\boldsymbol{w})||^2 + \frac{\beta}{2}||\nabla\ell_z(\boldsymbol{w})||^2,$$

which gives the desired result. □

We next derive the rate for each of the remaining feasible steps, that is, *SGD*, *ESGD* and $MAX2$.

A.5.5 SGD Subproblem

Lemma 11. *We assume that $\Omega = \mathbb{R}^d$, for every $z \in \mathcal{Z}$, $\ell_z(w)$ is β smooth and satisfies the RSI condition with constant μ. Let $\boldsymbol{w}^*$ be a solution of $f(\boldsymbol{w})$. We assume $\forall z \in \mathcal{Z}, \ell_{z_t^n}(\boldsymbol{w}^*) = 0$. Then, if we apply BORAT with $\eta \leq \hat{\eta} = \min\{\frac{1}{4\beta}, \frac{1}{4\mu}, \frac{\mu}{\beta^2}\}$ and we take the step resulting from the SGD subproblem for all t we have:*

$$\mathbb{E}[||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2] \leq (1 - \hat{\eta}\mu)||\boldsymbol{w}_t - \boldsymbol{w}^*||^2.$$

Proof.

$$||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2 \leq ||\Pi_\Omega(\boldsymbol{w}_t - \eta \boldsymbol{g}'_{z_t}) - \boldsymbol{w}^*||^2, \tag{A.95}$$

$$\leq ||\boldsymbol{w}_t - \eta \boldsymbol{g}_{z_t} - \boldsymbol{w}^*||^2, \tag{A.96}$$

$$= ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2 ||\boldsymbol{g}_{z_t}||^2 - 2\eta \langle \boldsymbol{g}_{z_t}, \boldsymbol{w}_t - \boldsymbol{w}^* \rangle, \tag{A.97}$$

$$\leq ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2 ||\boldsymbol{g}_{z_t}||^2 - 2\eta\mu ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2. \tag{A.98}$$

We have $||\boldsymbol{g}_{z_t}||^2 \leq 2\beta \ell_{z_t^n}(\boldsymbol{w}_t)$ from Lemma 3 and $\ell_{z_t^n}(\boldsymbol{w}_t) \leq \frac{\beta}{2}||\boldsymbol{w}_t - \boldsymbol{w}^*||^2$ from smoothness giving $||\boldsymbol{g}_{z_t}||^2 \leq \beta^2 ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2$. We can now upper bound the right hand side producing:

$$||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2 \leq ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2\beta^2 ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 - 2\eta\mu ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2, \tag{A.99}$$

$$||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2 \leq (1 - 2\eta\mu + \eta^2\beta^2) ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2, \tag{A.100}$$

$$||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2 \leq \left(1 - \eta(2\mu - \eta\beta^2)\right) ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2. \tag{A.101}$$

Now, if we select $\eta \leq \hat{\eta} = \min\{\frac{1}{2\beta}, \frac{1}{4\mu}, \frac{\mu}{\beta^2}\}$ in the worst case we get:

$$||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2 \leq (1 - \hat{\eta}\mu) ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2. \tag{A.102}$$

Taking expectations with respect to z_t:

$$\mathbb{E}[||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2] \leq \mathbb{E}[(1 - \hat{\eta}\mu) ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2]. \tag{A.103}$$

Noting that $\boldsymbol{w}_t$ does not depend on z_t, and neither does $\boldsymbol{w}^*$ due to the interpolation property:

$$\mathbb{E}[||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2] \leq (1 - \hat{\eta}\mu) ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2. \tag{A.104}$$

□

A.5.6 ESGD Subproblem

Lemma 12. *Let $z \in \mathcal{Z}$. We assume that ℓ_z is β-smooth and non-negative on $\mathbb{R}^d$. Additionally, we assume that $\eta \leq \frac{1}{2\beta}$. If we define $\gamma_t \doteq \min\{1, \frac{\ell_{z_t^n}(\boldsymbol{w}_t)}{\eta||\boldsymbol{g}_{zt}||^2}\}$, then we have:*

$$\gamma_t = 1, \ \forall t$$

Proof. From our assumption on η and Lemma 3 we have:

$$\eta \leq \frac{1}{2\beta} \leq \frac{\ell_{z_t^n}(\boldsymbol{w}_t)}{||\boldsymbol{g}_{zt}||^2}.$$

Rearranging gives:

$$1 \leq \frac{\ell_{z_t^n}(\boldsymbol{w}_t)}{\eta||\boldsymbol{g}_{zt}||^2}.$$

Plugging into the the definition of γ_t, gives the desired result. □

Lemma 13. *We assume that $\Omega = \mathbb{R}^d$, for every $z \in \mathcal{Z}$, $\ell_z(w)$ is β and satisfies the RSI condition with constant μ. Let $\boldsymbol{w}^*$ be a solution of $f(\boldsymbol{w})$. We assume $\forall z \in \mathcal{Z}, \ell_{z_t^n}(\boldsymbol{w}^*) = 0$. Then, if we apply BORAT with $\eta \leq \hat{\eta} = \min\{\frac{1}{4\beta}, \frac{1}{4\mu}, \frac{\mu}{\beta^2}\}$ and we take the step resulting from the ESGD subproblem for all t we have:*

$$\mathbb{E}[||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2] \leq (1 - \hat{\eta}\mu)||\boldsymbol{w}_t - \boldsymbol{w}^*||^2.$$

Proof. This proof loosely follows work by Vaswani et al. (2019). We start by plugging the parameter update into the expression for the euclidean distance from the next

iterate to the optimal point.

$$||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2 \leq ||\Pi_\Omega(\boldsymbol{w}_t - \eta \boldsymbol{g}'_{z_t}) - \boldsymbol{w}^*||^2, \tag{A.105}$$

$$\leq ||\boldsymbol{w}_t - \eta \boldsymbol{g}'_{z_t} - \boldsymbol{w}^*||^2, \tag{A.106}$$

$$= ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2 ||\boldsymbol{g}'_{z_t}||^2 - 2\eta \langle \boldsymbol{g}'_{z_t}, \boldsymbol{w}_t - \boldsymbol{w}^* \rangle, \tag{A.107}$$

$$= ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2 ||\boldsymbol{g}'_{z_t}||^2 - 2\eta \langle \boldsymbol{g}'_{z_t}, \boldsymbol{w}'_t + \eta\gamma_t \boldsymbol{g}_{z_t} - \boldsymbol{w}^* \rangle, \tag{A.108}$$

$$= ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2 ||\boldsymbol{g}'_{z_t}||^2 - 2\eta \langle \boldsymbol{g}'_{z_t}, \boldsymbol{w}'_t + \eta \boldsymbol{g}_{z_t} - \boldsymbol{w}^* \rangle, \text{ (Lemma 12)} \tag{A.109}$$

$$= ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2 ||\boldsymbol{g}'_{z_t}||^2 - 2\eta \langle \boldsymbol{g}'_{z_t}, \boldsymbol{w}'_t - \boldsymbol{w}^* \rangle - 2\eta^2 \langle \boldsymbol{g}'_{z_t}, \boldsymbol{g}_{z_t} \rangle. \tag{A.110}$$

Using the RSI condition:

$$||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2 \leq ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2 ||\boldsymbol{g}'_{z_t}||^2 - 2\eta\mu ||\boldsymbol{w}'_t - \boldsymbol{w}^*||^2 - 2\eta^2 \langle \boldsymbol{g}'_{z_t}, \boldsymbol{g}_{z_t} \rangle. \tag{A.111}$$

Using Lemma (9) to upper bound $-||\boldsymbol{w}'_t - \boldsymbol{w}^*||^2$,

$$= ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2 ||\boldsymbol{g}'_{z_t}||^2 - \eta\mu ||\boldsymbol{w}^* - \boldsymbol{w}_t||^2 + 2\eta\mu ||\boldsymbol{w}_t - \boldsymbol{w}'_t||^2 - 2\eta^2 \langle \boldsymbol{g}'_{z_t}, \boldsymbol{g}_{z_t} \rangle, \tag{A.112}$$

$$= (1 - \eta\mu) ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2 ||\boldsymbol{g}'_{z_t}||^2 + 2\eta\mu ||\boldsymbol{w}_t - \boldsymbol{w}'_t||^2 - 2\eta^2 \langle \boldsymbol{g}'_{z_t}, \boldsymbol{g}_{z_t} \rangle, \tag{A.113}$$

$$= (1 - \eta\mu) ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2 ||\boldsymbol{g}'_{z_t} - \boldsymbol{g}_{z_t}||^2 - \eta^2 ||\boldsymbol{g}_{z_t}||^2 + 2\eta\mu ||\boldsymbol{w}_t - \boldsymbol{w}'_t||^2, \tag{A.114}$$

$$= (1 - \eta\mu) ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2 ||\boldsymbol{g}'_{z_t} - \boldsymbol{g}_{z_t}||^2 - \eta^2 ||\boldsymbol{g}_{z_t}||^2 + 2\eta^3\mu ||\boldsymbol{g}_{z_t}||^2, \tag{A.115}$$

$$\leq (1 - \eta\mu) ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2\beta^2 ||\boldsymbol{w}'_t - \boldsymbol{w}_t||^2 - \eta^2 ||\boldsymbol{g}_{z_t}||^2 + 2\eta^3\mu ||\boldsymbol{g}_{z_t}||^2, \text{(smoothness)} \tag{A.116}$$

$$= (1 - \eta\mu) ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^4\beta^2 ||\boldsymbol{g}_{z_t}||^2 - \eta^2 ||\boldsymbol{g}_{z_t}||^2 + 2\eta^3\mu ||\boldsymbol{g}_{z_t}||^2, \tag{A.117}$$

$$= (1 - \eta\mu) ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2 \left(\eta^2\beta^2 - 1 + 2\eta\mu\right) ||\boldsymbol{g}_{z_t}||^2, \tag{A.118}$$

Taking expectations with respect to z_t:

$$\mathbb{E}[||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2] = \mathbb{E}[(1 - \eta\mu)||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2 \left(\eta^2\beta^2 - 1 + 2\eta\mu\right) ||\boldsymbol{g}_{z_t}||^2, \quad \text{(A.119)}$$

Noting that $\boldsymbol{w}_t$ does not depend on z_t, and neither does $\boldsymbol{w}^*$ due to the interpolation property:

$$\mathbb{E}[||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2] = (1 - \eta\mu)||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2 \left(\eta^2\beta^2 - 1 + 2\eta\mu\right) \mathbb{E}[||\boldsymbol{g}_{z_t}||^2], \quad \text{(A.120)}$$

If we set $\eta \leq \hat{\eta} = \min\{\frac{1}{2\beta}, \frac{1}{4\mu}, \frac{\mu}{\beta^2}\}$ then we have $\eta^2\beta^2 \leq \frac{1}{4}$, $2\eta\mu \leq \frac{1}{2}$,hence $(\eta^2\beta^2 - 1 + 2\eta\mu) \leq 0$.

$$\mathbb{E}[||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2] \leq (1 - \eta\mu)||\boldsymbol{w}_t - \boldsymbol{w}^*||^2. \quad \text{(A.121)}$$

Hence, if we insert the chosen value for η then we have:

$$\mathbb{E}[||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2] \leq (1 - \hat{\eta}\mu)||\boldsymbol{w}_t - \boldsymbol{w}^*||^2. \quad \text{(A.122)}$$

□

A.5.7 MAX2 Subproblem

Lemma 14. *We assume that $\Omega = \mathbb{R}^d$, for every $z \in \mathcal{Z}$, $l_z(w)$ is β smooth and satisfies the RSI condition with constant μ. Let $\boldsymbol{w}^*$ be a solution of $f(\boldsymbol{w})$. We assume $\forall z \in \mathcal{Z}, l_{z_t}(\boldsymbol{w}^*) = 0$. Then, if we apply BORAT with $\eta \leq \hat{\eta} = \min\{\frac{1}{4\beta}, \frac{1}{4\mu}, \frac{\mu}{\beta^2}\}$ and we take the step resulting from the MAX2 subproblem for all t we have:*

$$\mathbb{E}[||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2] \leq \left(1 - \frac{3}{8}\hat{\eta}\mu\right)^t ||\boldsymbol{w}_0 - \boldsymbol{w}^*||^2.$$

Proof. Note we assume $\gamma_t = 1$ as proved in Lemma 12.

$$||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2 \leq ||\Pi_\Omega(\boldsymbol{w}_t - \eta\alpha^{1t}\boldsymbol{g}_{z_t} - \eta\alpha^{2t}\boldsymbol{g}'_{z_t}) - \boldsymbol{w}^*||^2, \tag{A.123}$$

$$\leq ||\boldsymbol{w}_t - \eta\alpha^{1t}\boldsymbol{g}_{z_t} - \eta\alpha^{2t}\boldsymbol{g}'_{z_t} - \boldsymbol{w}^*||^2, \tag{A.124}$$

$$\begin{aligned} &= ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2||\alpha^{1t}\boldsymbol{g}_{z_t} + \alpha^{2t}\boldsymbol{g}'_{z_t}||^2 \\ &- 2\eta\langle\alpha^{1t}\boldsymbol{g}_{z_t} + \alpha^{2t}\boldsymbol{g}'_{z_t}, \boldsymbol{w}_t - \boldsymbol{w}^*\rangle, \end{aligned} \tag{A.125}$$

$$\begin{aligned} &= ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2||\alpha^{1t}\boldsymbol{g}_{z_t} + \alpha^{2t}\boldsymbol{g}'_{z_t}||^2 \\ &- 2\eta\alpha^{1t}\langle\boldsymbol{g}_{z_t}, \boldsymbol{w}_t - \boldsymbol{w}^*\rangle - 2\eta\alpha^{2t}\langle\boldsymbol{g}'_{z_t}, \boldsymbol{w}_t - \boldsymbol{w}^*\rangle, \end{aligned} \tag{A.126}$$

$$\begin{aligned} &= ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2||\alpha^{1t}\boldsymbol{g}_{z_t} + \alpha^{2t}\boldsymbol{g}'_{z_t}||^2 - 2\eta\alpha^{1t}\langle\boldsymbol{g}_{z_t}, \boldsymbol{w}_t - \boldsymbol{w}^*\rangle \\ &- 2\eta\alpha^{2t}\langle\boldsymbol{g}'_{z_t}, \boldsymbol{w}'_t + \eta\boldsymbol{g}_{z_t} - \boldsymbol{w}^*\rangle, \end{aligned} \tag{A.127}$$

$$\begin{aligned} &= ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2||\alpha^{1t}\boldsymbol{g}_{z_t} + \alpha^{2t}\boldsymbol{g}'_{z_t}||^2 - 2\eta\alpha^{1t}\langle\boldsymbol{g}_{z_t}, \boldsymbol{w}_t - \boldsymbol{w}^*\rangle \\ &- 2\eta\alpha^{2t}\langle\boldsymbol{g}'_{z_t}, \boldsymbol{w}'_t - \boldsymbol{w}^*\rangle - 2\eta^2\alpha^{2t}\langle\boldsymbol{g}'_{z_t}, \boldsymbol{g}_{z_t}\rangle, \end{aligned} \tag{A.128}$$

$$\begin{aligned} &= ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2||\alpha^{1t}\boldsymbol{g}_{z_t} + \alpha^{2t}\boldsymbol{g}'_{z_t}||^2 - 2\eta^2\alpha^{2t}\langle\boldsymbol{g}'_{z_t}, \boldsymbol{g}_{z_t}\rangle \\ &- 2\eta\alpha^{1t}\langle\boldsymbol{g}_{z_t}, \boldsymbol{w}_t - \boldsymbol{w}^*\rangle - 2\eta\alpha^{2t}\langle\boldsymbol{g}'_{z_t}, \boldsymbol{w}'_t - \boldsymbol{w}^*\rangle. \end{aligned} \tag{A.129}$$

We now make use of $-\langle \boldsymbol{g}_{z_t}, \boldsymbol{w}_t - \boldsymbol{w}^* \rangle \leq -\mu ||\boldsymbol{w}^* - \boldsymbol{w}_t||^2$ (RSI condition),

$$\begin{aligned} ||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2 \leq ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2 ||\alpha^{1t} \boldsymbol{g}_{z_t} + \alpha^{2t} \boldsymbol{g}'_{z_t}||^2 \\ - 2\eta^2 \alpha^{2t} \langle \boldsymbol{g}'_{z_t}, \boldsymbol{g}_{z_t} \rangle - 2\eta \alpha^{1t} \mu ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 - 2\eta \langle \alpha^{2t} \boldsymbol{g}'_{z_t}, \boldsymbol{w}'_t - \boldsymbol{w}^* \rangle, \end{aligned} \tag{A.130}$$

$$\begin{aligned} ||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2 \leq (1 - 2\eta \alpha^{1t} \mu) ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2 ||\alpha^{1t} \boldsymbol{g}_{z_t} + \alpha^{2t} \boldsymbol{g}'_{z_t}||^2 \\ - 2\eta^2 \alpha^{2t} \langle \boldsymbol{g}'_{z_t}, \boldsymbol{g}_{z_t} \rangle - 2\eta \alpha^{2t} \langle \boldsymbol{g}'_{z_t}, \boldsymbol{w}'_t - \boldsymbol{w}^* \rangle. \end{aligned} \tag{A.131}$$

Similarly, using $-\langle \boldsymbol{g}'_{z_t}, \boldsymbol{w}'_t - \boldsymbol{w}^* \rangle \leq -\mu ||\boldsymbol{w}^* - \boldsymbol{w}'_t||^2$ (RSI condition),

$$\begin{aligned} ||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2 \leq (1 - 2\eta \alpha^{1t} \mu) ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2 ||\alpha^{1t} \boldsymbol{g}_{z_t} + \alpha^{2t} \boldsymbol{g}'_{z_t}||^2 \\ - 2\eta^2 \alpha^{2t} \langle \boldsymbol{g}'_{z_t}, \boldsymbol{g}_{z_t} \rangle - 2\eta \alpha^{2t} \mu ||\boldsymbol{w}'_t - \boldsymbol{w}^*||^2. \end{aligned} \tag{A.132}$$

We now upper bound $-||\boldsymbol{w}'_t - \boldsymbol{w}^*||^2$, using Lemma 9:

$$\begin{aligned} ||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2 \leq (1 - 2\eta \alpha^{1t} \mu) ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2 ||\alpha^{1t} \boldsymbol{g}_{z_t} + \alpha^{2t} \boldsymbol{g}'_{z_t}||^2 \\ - 2\eta^2 \alpha^{2t} \langle \boldsymbol{g}'_{z_t}, \boldsymbol{g}_{z_t} \rangle - \alpha^{2t} \eta \mu ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + 2\alpha^{2t} \eta \mu ||\boldsymbol{w}_t - \boldsymbol{w}'_t||^2. \end{aligned} \tag{A.133}$$

This gives the following general form:

$$\begin{aligned} ||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2 \leq (1 - 2\eta \alpha^{1t} \mu - \alpha^{2t} \eta \mu) ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2 ||\alpha^{1t} \boldsymbol{g}_{z_t} + \alpha^{2t} \boldsymbol{g}'_{z_t}||^2 \\ - 2\eta^2 \alpha^{2t} \langle \boldsymbol{g}'_{z_t}, \boldsymbol{g}_{z_t} \rangle + 2\alpha^{2t} \eta \mu ||\boldsymbol{w}_t - \boldsymbol{w}'_t||^2. \end{aligned} \tag{A.134}$$

We now use the inequality $||\boldsymbol{w}_t - \boldsymbol{w}'_t|| = \eta^2 ||\boldsymbol{g}_{z_t}|| \leq \eta^2 \beta^2 ||\boldsymbol{w}_t - \boldsymbol{w}^*||$ to upper bound

the final term, see SGD proof (Section A.5.5) for derivation of inequality:

$$\begin{aligned}||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2 \leq (1 - 2\eta\alpha^{1t}\mu - \alpha^{2t}\eta\mu)||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 + \eta^2||\alpha^{1t}\boldsymbol{g}_{z_t} + \alpha^{2t}\boldsymbol{g}'_{z_t}||^2 \\ - 2\eta^2\alpha^{2t}\langle \boldsymbol{g}'_{z_t}, \boldsymbol{g}_{z_t}\rangle + 2\eta^3\beta^2\mu\alpha^{2t}||\boldsymbol{w}_t - \boldsymbol{w}^*||,\end{aligned} \tag{A.135}$$

$$\begin{aligned}||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2 \leq (1 - 2\eta\alpha^{1t}\mu - \alpha^{2t}\eta\mu + 2\eta^3\alpha^{2t}\beta^2\mu)||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 \\ + \eta^2||\alpha^{1t}\boldsymbol{g}_{z_t} + \alpha^{2t}\boldsymbol{g}'_{z_t}||^2 - 2\eta^2\alpha^{2t}\langle \boldsymbol{g}'_{z_t}, \boldsymbol{g}_{z_t}\rangle.\end{aligned} \tag{A.136}$$

We now simplify the last two terms, starting with the first:

$$\eta^2||\alpha^{1t}\boldsymbol{g}_{z_t} + \alpha^{2t}\boldsymbol{g}'_{z_t}||^2 = \eta^2((\alpha^{1t})^2||\boldsymbol{g}_{z_t}||^2 + 2\alpha^{1t}\alpha^{2t}\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t}\rangle + (\alpha^{2t})^2||\boldsymbol{g}'_{z_t}||^2). \tag{A.137}$$

Plugging in the expressions for α^{1t}, α^{2t},$\gamma_t = 1$, grouping like terms and simplifying gives the following:

$$\begin{aligned}&\eta^2||\alpha^{1t}\boldsymbol{g}_{z_t} + \alpha^{2t}\boldsymbol{g}'_{z_t}||^2 \\ &= \frac{(l_{z_t}(\boldsymbol{w}'_t) - l_{z_t}(\boldsymbol{w}_t))^2 + 2\eta(l_{z_t}(\boldsymbol{w}'_t) - l_{z_t}(\boldsymbol{w}_t))\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t}\rangle + \eta^2||\boldsymbol{g}_{z_t}||^2||\boldsymbol{g}'_{z_t}||^2}{||\boldsymbol{g}_{z_t} - \boldsymbol{g}'_{z_t}||^2}\end{aligned} \tag{A.138}$$

Plugging in α^{2t} into the remaining term gives the following expressions:

$$-2\eta^2\alpha^{2t}\langle \boldsymbol{g}'_{z_t}, \boldsymbol{g}_{z_t}\rangle = \frac{-2\eta^2||\boldsymbol{g}_{z_t}||^2\langle \boldsymbol{g}'_{z_t}, \boldsymbol{g}_{z_t}\rangle - 2\eta(l_{z_t}(\boldsymbol{w}'_t) - l_{z_t}(\boldsymbol{w}_t))\langle \boldsymbol{g}'_{z_t}, \boldsymbol{g}_{z_t}\rangle}{||\boldsymbol{g}_{z_t} - \boldsymbol{g}'_{z_t}||^2}. \tag{A.139}$$

Putting these together,

$$\begin{aligned}&\eta^2||\alpha^{1t}\boldsymbol{g}_{z_t} + \alpha^{2t}\boldsymbol{g}'_{z_t}||^2 - 2\eta^2\alpha^{2t}\langle \boldsymbol{g}'_{z_t}, \boldsymbol{g}_{z_t}\rangle \\ &= \frac{(l_{z_t}(\boldsymbol{w}'_t) - l_{z_t}(\boldsymbol{w}_t))^2 + 2\eta(l_{z_t}(\boldsymbol{w}'_t) - l_{z_t}(\boldsymbol{w}_t))\langle \boldsymbol{g}_{z_t}, \boldsymbol{g}'_{z_t}\rangle}{||\boldsymbol{g}_{z_t} - \boldsymbol{g}'_{z_t}||^2} \\ &+ \frac{\eta^2||\boldsymbol{g}_{z_t}||^2||\boldsymbol{g}'_{z_t}||^2 - 2\eta^2||\boldsymbol{g}_{z_t}||^2\langle \boldsymbol{g}'_{z_t}, \boldsymbol{g}_{z_t}\rangle - 2\eta(l_{z_t}(\boldsymbol{w}'_t) - l_{z_t}(\boldsymbol{w}_t))\langle \boldsymbol{g}'_{z_t}, \boldsymbol{g}_{z_t}\rangle}{||\boldsymbol{g}_{z_t} - \boldsymbol{g}'_{z_t}||^2}.\end{aligned} \tag{A.140}$$

Cancelling terms gives,

$$\begin{aligned}&\eta^2||\alpha^{1t}\boldsymbol{g}_{z_t}+\alpha^{2t}\boldsymbol{g}'_{z_t}||^2-2\eta^2\alpha^{2t}\langle\boldsymbol{g}'_{z_t},\boldsymbol{g}_{z_t}\rangle\\&=\frac{(l_{z_t}(\boldsymbol{w}'_t)-l_{z_t}(\boldsymbol{w}_t))^2+\eta^2||\boldsymbol{g}_{z_t}||^2||\boldsymbol{g}'_{z_t}||^2-2\eta^2||\boldsymbol{g}_{z_t}||^2\langle\boldsymbol{g}'_{z_t},\boldsymbol{g}_{z_t}\rangle}{||\boldsymbol{g}_{z_t}-\boldsymbol{g}'_{z_t}||^2}.\end{aligned}\tag{A.141}$$

From $\alpha^{2t}\geq 0$ we have $\eta||\boldsymbol{g}_{z_t}||^2\geq l_{z_t}(\boldsymbol{w}_t)-l_{z_t}(\boldsymbol{w}'_t)$ hence we can upper bound $(l_{z_t}(\boldsymbol{w}'_t)-l_{z_t}(\boldsymbol{w}_t))^2$ by $\eta^2||\boldsymbol{g}'_{z_t}||^4$:

$$\begin{aligned}&\eta^2||\alpha^{1t}\boldsymbol{g}_{z_t}+\alpha^{2t}\boldsymbol{g}'_{z_t}||^2-2\eta^2\alpha^{2t}\langle\boldsymbol{g}'_{z_t},\boldsymbol{g}_{z_t}\rangle\\&\leq\frac{\eta^2||\boldsymbol{g}'_{z_t}||^4+\eta^2||\boldsymbol{g}_{z_t}||^2||\boldsymbol{g}'_{z_t}||^2-2\eta^2||\boldsymbol{g}_{z_t}||^2\langle\boldsymbol{g}'_{z_t},\boldsymbol{g}_{z_t}\rangle}{||\boldsymbol{g}_{z_t}-\boldsymbol{g}'_{z_t}||^2}\leq\eta^2||\boldsymbol{g}_{z_t}||^2.\end{aligned}\tag{A.142}$$

Again, we use the inequality $||\boldsymbol{w}_t-\boldsymbol{w}'_t||=\eta^2||\boldsymbol{g}_{z_t}||\leq\eta^2\beta^2||\boldsymbol{w}_t-\boldsymbol{w}^*||$:

$$\eta^2||\alpha^{1t}\boldsymbol{g}_{z_t}+\alpha^{2t}\boldsymbol{g}'_{z_t}||^2-2\eta^2\alpha^{2t}\langle\boldsymbol{g}'_{z_t},\boldsymbol{g}_{z_t}\rangle\leq\eta^2||\boldsymbol{g}_{z_t}||^2\leq\eta^2\beta^2||\boldsymbol{w}_{t+1}-\boldsymbol{w}^*||^2\tag{A.143}$$

Hence, we get the following expression:

$$||\boldsymbol{w}_{t+1}-\boldsymbol{w}^*||^2\leq(1-2\eta\alpha^{1t}\mu-\alpha^{2t}\eta\mu+2\eta^3\alpha^{2t}\beta^2\mu)||\boldsymbol{w}_t-\boldsymbol{w}^*||^2+\eta^2\beta^2||\boldsymbol{w}_{t+1}-\boldsymbol{w}^*||^2\rangle,\tag{A.144}$$

$$||\boldsymbol{w}_{t+1}-\boldsymbol{w}^*||^2\leq(1-2\eta\alpha^{1t}\mu-\alpha^{2t}\eta\mu+2\eta^3\alpha^{2t}\beta^2\mu+\eta^2\beta^2)||\boldsymbol{w}_t-\boldsymbol{w}^*||^2,\tag{A.145}$$

$$||\boldsymbol{w}_{t+1}-\boldsymbol{w}^*||^2\leq(1-\eta\mu-\alpha^{1t}\eta\mu+2\eta^3\beta^2\mu-2\eta^3\alpha^{1t}\beta^2\mu+\eta^2\beta^2)||\boldsymbol{w}_t-\boldsymbol{w}^*||^2.\tag{A.146}$$

Upper bounding $-\alpha^{1t}$ by 0,

$$||\boldsymbol{w}_{t+1}-\boldsymbol{w}^*||^2\leq(1-\eta\mu+2\eta^3\beta^2\mu+\eta^2\beta^2)||\boldsymbol{w}_t-\boldsymbol{w}^*||^2.\tag{A.147}$$

Taking expectations with respect to z_t:

$$\mathbb{E}[||\boldsymbol{w}_{t+1}-\boldsymbol{w}^*||^2]\leq\mathbb{E}[(1-\eta\mu+2\eta^3\beta^2\mu+\eta^2\beta^2)||\boldsymbol{w}_t-\boldsymbol{w}^*||^2].\tag{A.148}$$

Noting that $\boldsymbol{w}_k$ does not depend on z_t, and neither does $\boldsymbol{w}^*$ due to the interpolation property:

$$\mathbb{E}[||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2] \leq (1 - \eta\mu + 2\eta^3\beta^2\mu + \eta^2\beta^2)||\boldsymbol{w}_t - \boldsymbol{w}^*||^2. \tag{A.149}$$

For this step to be convergent we need the following condition to hold $2\eta^3\beta^2\mu + \eta^2\beta^2 - \eta\mu \leq 0$. However, for $\eta \leq \hat{\eta} = \min\{\frac{1}{4\beta}, \frac{1}{4\mu}, \frac{\mu}{2\beta^2}\}$ we have:

$$2\eta^3\beta^2\mu + \eta^2\beta^2 - \eta\mu \leq \frac{1}{8}\eta\mu + \frac{1}{2}\eta\mu - \eta\mu \leq -\frac{3}{8}\eta\mu. \tag{A.150}$$

Hence, we recover the rate:

$$\mathbb{E}[||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2] \leq \left(1 - \frac{3}{8}\hat{\eta}\mu\right)^t ||\boldsymbol{w}_0 - \boldsymbol{w}^*||^2. \tag{A.151}$$

□

A.5.8 Worst Case Rate

It is clear by inspection that the worst case rate derived corresponds to the MAX2 subproblem. Hence, in the worst case this step is taken for all t, and thus a trivial induction gives the result of Theorem 2.

Appendix B

Appendix: Additional Results for Chapter 4

B.1 Empirical Run Time

In this appendix we detail the effect of increasing N on the run time of BORAT. Due to each update requiring $N-1$ gradient evaluations BORAT with $N \geq 2$ takes significantly longer between updates than other methods. However, BORAT achieves good empirical convergence rates requiring $N-1$ fewer parameter updates than other methods, as shown in the results section. Hence, we consider the epoch time and show that BORAT has a similar run time to SGD for each pass through the data.

Increasing N both increases the run time of Algorithm 1 and BORAT must compute extra dot products when calculating Q. A naive implementation of Algorithm 1 has a time complexity of $\mathcal{O}\left(\sum_{k=1}^{N} \frac{N!}{k!(N-k)!} k^3\right)$. However, if we exploit the parallel nature of this algorithm where the sub problems are solved simultaneously, the time complexity reduces to $\mathcal{O}\left(N^3\right)$ as discussed in Section 4.3.5. Additionally, we need only run Algorithm 1 once every $N-1$ batches so the per epoch time complexity is $\mathcal{O}\left(N^2\right)$. In practice, Algorithm 3 is only responsible for a small fraction of the run

time, where its contribution is determined by the relative size of the model and N.

Table B.1 shows the effect of this extra computation on the training epoch time is not significant and the time complexity with a parallel implementation scales approximately linearly with N. Moreover, Table B.1 shows for large learning problems, such as ImageNet the extra run time when increasing N is negligible.

Optimiser	SGD	BORAT	BORAT	BORAT	BORAT
N	1	2	3	4	5
Time (s)	51.0	55.6	68.2	69.2	74.3
Optimiser	BORAT	BORAT	BORAT	BORAT	BORAT
N	6	7	8	9	10
Time (s)	77.8	82.8	88.7	94.1	99.5

Table B.1: *Average training epoch time for CIFAR-100 data set, shown for varying N. Time quoted using a batch size of 128, CIFAR-100, CE loss, a Wide ResNet 40-4, and a parallel implementation of BORAT. All Optimiser had access to 3 CPU cores, and one TITAN Xp GPU.*

Optimiser	BORAT	BORAT	BORAT
N	2	3	5
Time (s)	885.50	910.49	934.79

Table B.2: *Average BORAT training epoch time for ImageNet data set, shown for varying N. Time quoted using a batch size of 1024, ImageNet, CE loss, a ResNet18, and a parallel implantation of BORAT. All optimisers had access to 12 CPU cores, and 4 TITAN Xp GPUs.*

B.2 Robustness of Adam Optimiser

In Figure B.1 we include results when using Adam for the experiments without label noise of Section 4.5.2.

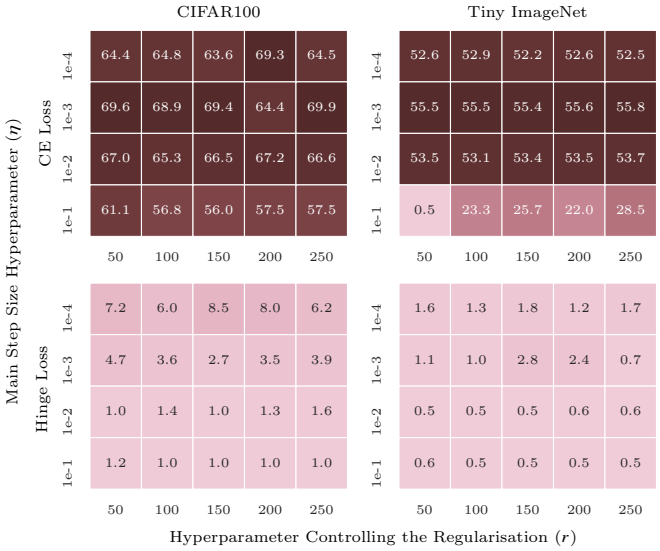


Figure B.1: *Adam's robustness to hyperparameters for the CE and multi-class hinge losses on the CIFAR-100 and Tiny ImageNet data sets. Colour represents test performance, where darker colours correspond to higher values. When training on the CIFAR-100 data set with CE loss Adam is robust to its hyperparameters. However, its peak performance is 5% less than ALI-G or BORAT. On the Tiny ImageNet data set Adam produces good results for* $\eta \leq 0.1$*, but again its peak performance lags roughly 2% behind ALI-G and BORAT. In combination with the multi-class hinge loss Adam does not produce good results for either data set. BORAT offers better peak performance than Adam, and similar robustness when using the CE loss. BORAT performs significantly better for the multi-class hinge loss.*

B.3 CIFAR Hyperparameters and Variance

In Table B.3 we detail the hyperparameters and variance for the ALI-G and BORAT results reported in Table 4.3. For other optimisation methods please refer to Appendix E of Berrada et al. (2020).

Table B.3: *CIFAR Hyperparameters (BORAT ALI-G)*

Data Set	Model	Hyperparameters				Test Accuracy	
		N	η	r	Batch Size	Mean	StD
CIFAR-10	WRN	2	0.1	50	128	95.4	0.13
		3	1	100	128	95.4	0.05
		5	1	75	128	95.0	0.08
	DN	2	0.1	100	64	94.5	0.09
		3	1	75	256	94.9	0.13
		5	1	75	128	94.9	0.13
CIFAR-100	WRN	2	0.1	50	512	76.1	0.21
		3	0.1	50	256	76.0	0.16
		5	0.1	50	128	75.8	0.22
	DN	2	0.1	75	256	76.2	0.14
		3	0.1	75	128	76.5	0.38
		5	0.1	75	64	75.7	0.03

Appendix C

Appendix: Proofs of Theorems in Chapter 5

C.1 Theoretical Results

Theorem 3 (Worst Case Bound - Smooth and Convex)**.** *We assume that for every $z \in \mathcal{Z}$, ℓ_z is convex and β-smooth. Let $\boldsymbol{w}_\star$ be a solution of $(\mathcal{P})$ such that $\forall z \in \mathcal{Z}$, $\ell_z(\boldsymbol{w}_\star) \leq \varepsilon$. Further assume that $\eta \leq \frac{1}{2\beta}$ and $\lambda = 0$. Then if we apply ALI-G+ with a maximal learning rate of η to f, we have:*

$$\min_t f(\boldsymbol{w}_t) - f_\star \leq \frac{\|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2}{\eta(\frac{T}{K} + 1)} + \epsilon. \tag{5.3}$$

Proof. This proof is based on Theorem 6 in the appendix of Berrada et al. (2020). In order to derive this worst case bound we consider the performance of ALI-G+ before the first AOV update when $\tilde{\ell}_z^k = 0, \forall z \in \mathcal{Z}$. We start from the definition of $\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|$:

$$\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_t - \gamma_t \nabla \ell_{z_t}(\boldsymbol{w}_t) - \boldsymbol{w}_\star\|^2. \tag{C.1}$$

Expanding:

$$\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 - 2\gamma_t \nabla \ell_{z_t}(\boldsymbol{w}_t)^\intercal (\boldsymbol{w}_t - \boldsymbol{w}_\star) + \gamma_t^2 \|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2. \quad \text{(C.2)}$$

From convexity we have:

$$\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 - 2\gamma_t (\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\boldsymbol{w}_\star)) + \gamma_t^2 \|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2. \quad \text{(C.3)}$$

Before the first AOV update we have $\tilde{\ell}_z^k = 0, \forall z \in \mathcal{Z}$ and we have as:

$$\gamma_t \triangleq \max \left\{ \min \left\{ \eta, \frac{\ell_{z_t}(\boldsymbol{w}_t) - \tilde{\ell}_z^k}{\|\nabla \ell_{z_t}\|^2} \right\}, 0 \right\}, \quad \text{(C.4)}$$

$$\gamma_t = \max \left\{ \min \left\{ \eta, \frac{\ell_{z_t}(\boldsymbol{w}_t) - 0}{\|\nabla \ell_{z_t}\|^2} \right\}, 0 \right\} = \min \left\{ \eta, \frac{\ell_{z_t}(\boldsymbol{w}_t)}{\|\nabla \ell_{z_t}\|^2} \right\}. \quad \text{(C.5)}$$

Hence it is possible to use the upper bound $\gamma_t \leq \frac{\ell_{z_t}(\boldsymbol{w}_t)}{\|\nabla \ell_{z_t}\|^2}$ in Equation (C.3) by:

$$\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 - 2\gamma_t (\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\boldsymbol{w}_\star)) + \gamma_t^2 \|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2, \quad \text{(C.6)}$$

$$\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 - 2\gamma_t (\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\boldsymbol{w}_\star)) + \gamma_t \ell_{z_t}(\boldsymbol{w}_t), \quad \text{(C.7)}$$

$$\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 - \gamma_t (\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\boldsymbol{w}_\star)) + \gamma_t \ell_{z_t}(\boldsymbol{w}_\star). \quad \text{(C.8)}$$

We now turn out attention to upper bounding $-\gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\boldsymbol{w}_\star))$ in Equation (C.8). We do this by considering two cases corresponding to if $\gamma_t = \frac{\ell_{z_t}(\boldsymbol{w}_t)}{\|\nabla \ell_{z_t}\|^2}$ or $\gamma_t = \eta$. Starting with the first case, we have $\gamma_t \leq \eta$ from Equation (C.5). However, we also have:

$$\gamma_t = \frac{\ell_{z_t}(\boldsymbol{w}_t)}{\|\nabla \ell_{z_t}\|^2} \geq \frac{1}{2\beta} \geq \eta, \quad \text{(C.9)}$$

where the first inequality is from Lemma 3 and the second comes from our assumption on the step size. As we have $\gamma_t \geq \eta$ and $\gamma_t \leq \eta$ we can conclude that $\gamma_t = \eta$.

For the second case, we recover $\gamma_t = \eta$ by definition. We propose the upper bound:

$$-\gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\boldsymbol{w}_\star)) \leq -\eta(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\boldsymbol{w}_\star)). \tag{C.10}$$

If $\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\boldsymbol{w}_\star)) \leq 0$ this bound holds by the fact that $\gamma_t \leq \eta$. If however $\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\boldsymbol{w}_\star)) \geq 0$ this bound holds in equality as $\gamma_t = \eta$. Plugging (C.10) into (C.8) gives:

$$\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 - \eta(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\boldsymbol{w}_\star)) + \gamma_t \ell_{z_t}(\boldsymbol{w}_\star). \tag{C.11}$$

We next use upper bound $\gamma_t \ell_{z_t}(\boldsymbol{w}_\star) \leq \eta\epsilon$ to upper bound (C.11). This upper bound holds as we know $\ell_{z_t}(\boldsymbol{w}_\star) \geq 0$ from our assumption on a non-negativity of ℓ and $\gamma_t \geq \eta$. Thus, we arrive at:

$$\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 - \eta(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\boldsymbol{w}_\star)) + \eta\epsilon. \tag{C.12}$$

By taking the expectation and using a telescopic sum, over the T/K steps before the first AOV update we obtain:

$$0 \leq \|\boldsymbol{w}_{T+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2 - \sum_{t=0}^{T/K} \left(\eta(\mathbb{E}[f(\boldsymbol{w}_t)] - f_\star) + \eta\epsilon\right). \tag{C.13}$$

Rearranging we finally obtain:

$$\min_t f(\boldsymbol{w}_t) - f_\star \leq \frac{1}{T+1} \sum_{t=0}^{T/K} f(\boldsymbol{w}_t) - f_\star \leq \frac{\|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2}{\eta(\frac{T}{K} + 1)} + \epsilon. \tag{C.14}$$

□

C.2 Strongly Convex

In this appendix we provide the following bound on the worst case loss for ALI-G+ applied to smooth and strongly convex settings.

Theorem 6 (Worst Case Bound - Smooth and Strongly Convex)**.** *We assume that for every $z \in \mathcal{Z}$, ℓ_z is α-strongly convex and β-smooth. Let $\boldsymbol{w}_\star$ be a solution of ($\mathcal{P}$) such that $\forall z \in \mathcal{Z}$, $\ell_z(\boldsymbol{w}_\star) \leq \epsilon$. Further, assume that $\eta \leq \frac{1}{2\beta}$ and $\lambda = 0$. Then, if we apply ALI-G+ with a maximal learning-rate of η to f, we have:*

$$\min_{t \in T} f(\boldsymbol{w}_t) - f_\star \leq \frac{\beta}{2} \exp\left(1 - \frac{\alpha \eta T}{2K}\right) \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2 + \frac{\beta \epsilon}{\alpha}. \tag{C.15}$$

Proof. This proof is based on Theorem 8 in the appendix of Berrada et al. (2020). In order to derive this worst case bound we consider the performance of ALI-G+ before the first AOV update when $\tilde{\ell}_z^k = 0, \forall z \in \mathcal{Z}$. We start from equation (C.8):

$$\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2 \leq \|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 - \eta(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\boldsymbol{w}_\star)) + \eta\epsilon \tag{C.16}$$

Taking the expectation over $z_t | z_{t-1}$, we obtain:

$$\mathbb{E}_{z_t | z_{t-1}}[\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2] \leq \|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 - \eta(f(\boldsymbol{w}_t) - f(\boldsymbol{w}_\star)) + \eta\epsilon. \tag{C.17}$$

Therefore, using Lemma 4 we can write:

$$\mathbb{E}_{z_t | z_{t-1}}[\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_\star\|^2] \leq \|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 - \frac{\alpha\eta}{2}\|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 + \eta\epsilon, \tag{C.18}$$

$$= \left(1 - \frac{\alpha\eta}{2}\right) \|\boldsymbol{w}_t - \boldsymbol{w}_\star\|^2 + \eta\epsilon. \tag{C.19}$$

A simple induction over the first $\frac{T}{K}$ before the first AOV update gives that:

$$\mathbb{E}[\|\boldsymbol{w}_{\frac{T}{K}+1} - \boldsymbol{w}_\star\|^2] \leq \left(1 - \frac{\alpha\eta}{2}\right)^{\frac{T}{K}} \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2 + \eta\epsilon \sum_{t=0}^{T/K} \left(1 - \frac{\alpha\eta}{2}\right)^t, \tag{C.20}$$

$$\leq \left(1 - \frac{\alpha\eta}{2}\right)^{\frac{T}{K}} \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2 + \eta\epsilon \sum_{t=0}^{\infty} \left(1 - \frac{\alpha\eta}{2}\right)^t, \tag{C.21}$$

$$= \left(1 - \frac{\alpha\eta}{2}\right)^{\frac{T}{K}} \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2 + \frac{\eta\epsilon}{1 - \left(1 - \frac{\alpha\eta}{2}\right)}, \tag{C.22}$$

$$= \left(1 - \frac{\alpha\eta}{2}\right)^{\frac{T}{K}} \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2 + \frac{2\epsilon}{\alpha}. \tag{C.23}$$

Let $\boldsymbol{w} \in \mathbb{R}^d$. Then using Lemma 3.4 of Bubeck (2015), we have:

$$\forall\, \boldsymbol{w} \in \mathbb{R}^d,\ \ |f(\boldsymbol{w}) - f(\boldsymbol{w}_\star) - \nabla f(\boldsymbol{w}_\star)^\intercal(\boldsymbol{w} - \boldsymbol{w}_\star)| \leq \frac{\beta}{2}\|\boldsymbol{w} - \boldsymbol{w}_\star\|^2. \tag{C.24}$$

Noting that $\nabla f(\boldsymbol{w}_\star) = 0$ and $f(\boldsymbol{w}) \geq f(\boldsymbol{w}_\star)$ we can write:

$$\forall\, \boldsymbol{w} \in \mathbb{R}^d,\ \ f(\boldsymbol{w}) - f(\boldsymbol{w}_\star) \leq \frac{\beta}{2}\|\boldsymbol{w} - \boldsymbol{w}_\star\|^2. \tag{C.25}$$

Setting $\boldsymbol{w} = \boldsymbol{w}_{\frac{T}{K}+1}$ and taking expectations with respect to z_t gives:

$$\min_{t \in T} f(\boldsymbol{w}_t) - f_\star \leq \mathbb{E}[f(\boldsymbol{w}_{\frac{T}{K}+1})] - f(\boldsymbol{w}_\star) \leq \frac{\beta}{2}\mathbb{E}[\|\boldsymbol{w}_{\frac{T}{K}+1} - \boldsymbol{w}_\star\|^2]. \tag{C.26}$$

Putting (C.23) and (C.26) together gives the desired result:

$$\min_{t \in T} f(\boldsymbol{w}_t) - f_\star \leq \frac{\beta}{2}\mathbb{E}[\|\boldsymbol{w}_{TK+1} - \boldsymbol{w}_\star\|^2], \tag{C.27}$$

$$\leq \frac{\beta}{2}\left(1 - \frac{\alpha\eta}{2}\right)^{\frac{T}{K}} \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2 + \frac{\beta\epsilon}{\alpha}, \tag{C.28}$$

$$\leq \frac{\beta}{2}\exp\left(1 - \frac{\alpha\eta T}{2k}\right) \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2 + \frac{\beta\epsilon}{\alpha}, \tag{C.29}$$

$$\leq \frac{\beta}{2}\exp\left(1 - \frac{\alpha\eta T}{2K}\right) \|\boldsymbol{w}_0 - \boldsymbol{w}_\star\|^2 + \frac{\beta\epsilon}{\alpha}. \tag{C.30}$$

□

Appendix D

Appendix: Additional Results for Chapter 5

D.1 Unsuccessful Approaches

In this appendix we introduce some unsuccessful ideas for Algorithms 3 and 4.

Parameter Updates An idea we experimented with for Algorithm 3 was defining an augmented loss function ℓ'_z as the pointwise maximum of the z^{th} loss function and its relevant AOV, specifically:

$$\ell'_z(\boldsymbol{w}) \triangleq \max\left\{\ell_{z_t}(\boldsymbol{w}_t), \tilde{\ell}_z^k\right\}. \tag{D.1}$$

This augmented loss function would then be used in the following parameter update:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \gamma_t \mathbb{E}_{z_t \in \mathcal{B}_t}[\nabla_{\boldsymbol{w}} \ell'_{z_t}(\boldsymbol{w}_t)], \tag{D.2}$$

$$\gamma_t \triangleq \max\left\{\min\left\{\eta, \frac{\mathbb{E}_{z_t \in \mathcal{B}_t}[\ell'_{z_t}(\boldsymbol{w}) - \tilde{\ell}_z^k]}{\|\mathbb{E}_{z_t \in \mathcal{B}_t}[\nabla_{\boldsymbol{w}} \ell'_{z_t}]\|^2}\right\}, 0\right\}, \tag{D.3}$$

where $\mathcal{B}_t$ is the set of indices z_t of the example selected within the batch at time t. This formulation excluded loss functions ℓ_z that had reached their AOV both in the

step size calculation and the descent direction. This had the effect of focusing on examples that had not yet reached their AOV. While this might sound desirable at first, in fact it is counterproductive in the non-interpolation setting. In this setting it is impossible to achieve zero loss on all samples simultaneously. Hence, focusing on the hardest samples is detrimental when trying to minimise the mean loss.

AOV Updates The AOV increase in lines 9-10 of Algorithm 4 was inspired by Hazan and Kakade (2022) and we did not try alternate schemes. For the AOV decrease in lines 6-7 of Algorithm 4 we initially tried backtracking to the previous AOV that had been reached rather than half way. We found this worked slightly worse in practice, as it resulted in the AOVs oscillating more.

D.2 Additional Plots

In this section we provide a variety of training curves produced by ALI-G+ in a number of settings. We start by showing ALI-G+ 's performance on the SVHN data set where interpolation holds, see Figure D.1. The next few plots detail runs of ALI-G+ on the slightly more challenging CIFAR-100 data set. In Figure D.2, in contrast to 5.4 we show the behaviour of ALI-G+ both with and without data augmentation. In Figure D.3 we show training curves when $K = 10$ to highlight why $K = 5$ is preferred. Finally, we show the behaviour when using ALI-G+ to train a ResNet18 (He et al., 2016) on the ImageNet data set in Figure D.4.

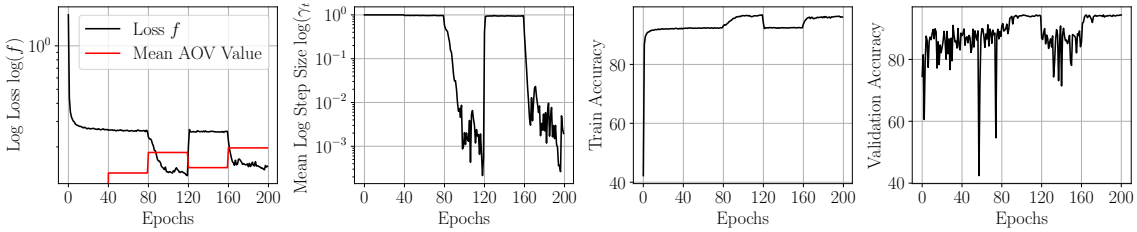


Figure D.1: *Curves produced by training a small ResNet on the SVHN data set (Netzer et al., 2011) with the ALI-G+ optimiser. These results were produced with* $\eta = 1.0, \lambda = 10^{-3}$. *No data augmentation was used as is standard for SVHN. The AOVs were updated every 40 epochs. The maximum step size* η *is selected for the first 80 epochs. At epoch 80 the AOVs are increased to a point where the mean step size decreases. This was followed by a sharp decrease in loss value over the next 20 epochs. This, in turn, results in the mean step size dropping further and becoming zero for many batches. At epoch 120 the mean AOV value was significantly higher than the mean loss value* $\ell_z(\bar{\boldsymbol{w}}_k)$ *resulting in the majority of AOVs being decreased in value during the update. The updated AOV values resulted in the maximum step size being selected again for most batches. This causes the loss to increase sharply. Finally, at epoch 160 the AOVs are increased again resulting in a similar behaviour to that at epoch 80.*

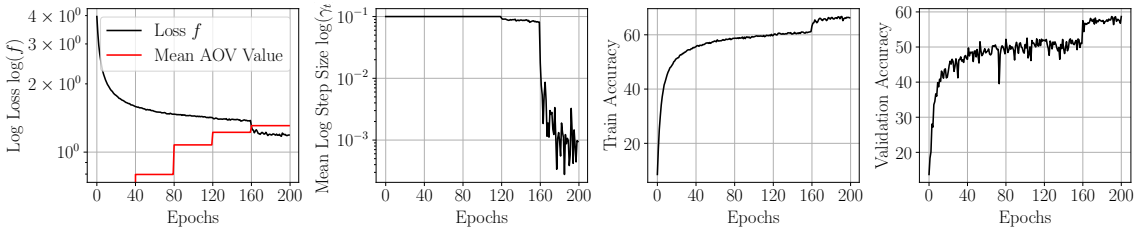


Figure D.2: *Curves produced by training a small ResNet on the CIFAR-100 data set (Krizhevsky, 2009) with data augmentation with the ALI-G+ optimiser. These results were produced with* $\eta = 0.1, \lambda = 10^{-3}$. *The AOVs are updated every 40 epochs. The maximum step size* η *is selected for the majority of batches during the first 120 epochs. For the remaining 140 epochs the step size was tailored to each batch.*

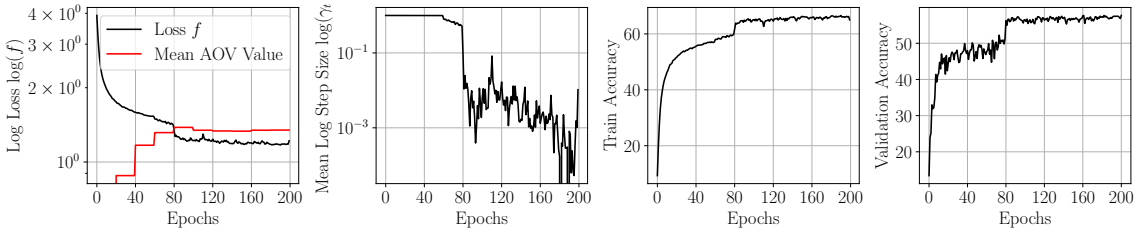


Figure D.3: *Curves produced by training a small ResNet on the CIFAR-100 data set (Krizhevsky, 2009), again with data augmentation with the ALI-G+ optimiser, however, we set* $K = 10$. *These results were produced with* $\eta = 1.0, \lambda = 10^{-4}$. *The AOVs are updated every 20 epochs. The maximum step size* η *is selected for the majority of batches during the first 60 epochs. For the last 140 epochs the step size was tailored to each batch. However, the accuracy does not improve significantly during the last half of training. Due to the rapid AOV updates the mean AOV stabilises at a suboptimally high value. This results in a small step size being used on average, and thus little progress is made for the remainder of the training period. This results in* $K = 10$ *achieving slightly worse accuracy than* $K = 5$.

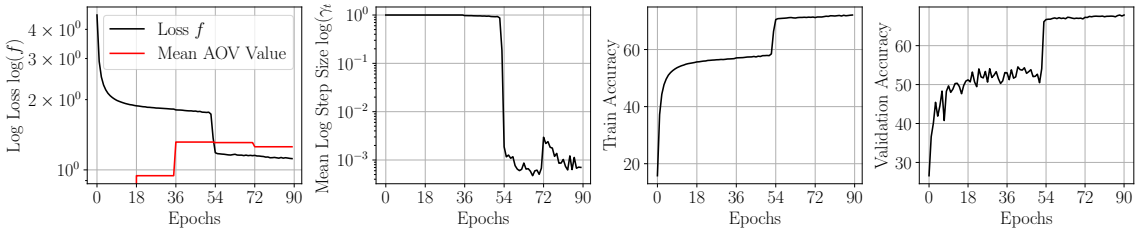


Figure D.4: *Curves produced by training a ResNet18 on the ImageNet data set (Deng et al., 2009) with data augmentation with the ALI-G+ optimiser. These results were produced with* $\eta = 1.0, \lambda = 10^{-4}$. *The AOVs are updated every 18 epochs. The maximum step size* η *is selected for the majority of batches during the first 36 epochs. For the remaining 54 epochs ALI-G+ tailors the step size to each batch.*

D.3 Additional Results

In this appendix we provide results for modified versions of ALI-G+ applied to many of the image classification tasks detailed in Section 5.4.2 within the body of the thesis. We state results for three additional values of K and for two version of ALI-G+ that use a global step size instead of a per sample step size. These step sizes are detailed below. The global AOV $\tilde{f}^k$ that aims to approximate $f_\star$ is updated using the equations in Algorithm 4 with the the pointwise estimates $\ell_{z_t}(\bar{\boldsymbol{w}})$ replaced with a global version $\tilde{f}(\bar{\boldsymbol{w}})$. Here we define $\mathcal{B}_t$ as the set of indices z_t within the batch at time t.

Global ALI-G+ variant 1 (GALIG1+)

$$\gamma_t \triangleq \max\left\{\min\left\{\eta, \frac{|\mathcal{B}_t| \sum_{z_t \in \mathcal{B}_t}[f(\boldsymbol{w}_t) - \tilde{f}^k]}{\| \sum_{z_t \in \mathcal{B}_t}[\nabla_{\boldsymbol{w}} \ell_{z_t}]\|^2}\right\}, 0\right\}. \tag{D.4}$$

Global ALI-G+ variant 2 (GALIG2+)

$$\gamma_t \triangleq \max\left\{\min\left\{\eta, \frac{|\mathcal{B}_t| \sum_{z_t \in \mathcal{B}_t}[\ell_{z_t}(\boldsymbol{w}_t) - \tilde{f}^k]}{\| \sum_{z_t \in \mathcal{B}_t}[\nabla_{\boldsymbol{w}} \ell_{z_t}]\|^2}\right\}, 0\right\}. \tag{D.5}$$

Results Table D.1 details the results of these experiments. ALI-G+ performs best for $K \in \{5, 10\}$, with the accuracies falling away as K is increased or decreased outside this range. GALIG1+ performs similar to ALI-G+ , which indicates that using a single global AOV could be sufficient in these settings. This could be a promising direction for future work that would reduce the memory footprint. GALIG2+ performs slightly worse than both ALI-G+ and GALIG1+. This suggests in the step size mixing batch-wise and global estimates of the loss should be avoided, however, further investigation is needed.

	SVHN	CIFAR-10		CIFAR-100		Tiny ImageNet	
	Test Acc (%)					Val Acc (%)	
Data Aug	No	No	Yes	No	Yes	No	Yes
ALI-G($K \approx 1$)	93.7	80.8	86.2	47.5	57.9	35.6	41.8
ALI-G+ ($K = 3$)	95.4	84.8	86.5	54.6	57.4	36.1	41.4
ALI-G+ ($K = 5$)	**95.5**	**85.0**	**87.2**	56.1	**59.4**	39.8	**42.6**
ALI-G+ ($K = 10$)	95.0	**85.0**	86.8	**56.6**	58.0	**39.9**	42.3
ALI-G+ ($K = 20$)	94.8	83.8	85.9	54.5	56.0	38.9	40.8
ALI-G+ ($K = 5$)	**95.5**	**85.0**	87.2	56.1	**59.4**	39.8	**42.6**
GALIG1+ ($K = 5$)	**95.5**	84.4	**87.3**	**56.2**	**59.4**	**40.4**	**42.6**
GALIG2+ ($K = 5$)	94.8	84.5	86.3	56.0	56.9	35.5	41.3

Table D.1: *Accuracies for ALI-G ($K \approx 1$) and ALI-G+ with $K \in \{3, 5, 10, 20\}$ on a selection of standard image classification data sets. ALI-G+ with $K = 10$ offers comparable results to $K = 5$, however, when K is increased to $K = 20$ the performance becomes noticeably worse. We also show two results for two modified versions of ALI-G+ with a global step size.*

D.4 Cross-Validation Hyperparameters

Here we specify the exact hyperparameters considered to generate Table 5.1.

Optimiser	Step Size	Regularisation
SGD_{Step}	$\eta_0 \in \{0.1, 0.01\}$	wd $\in \{0, 10^{-5}, 10^{-4}, 10^{-3}\}$
SGD_{Const}	$\eta \in \{0.1, 0.01\}$	wd $\in \{0, 10^{-5}, 10^{-4}, 10^{-3}\}$
ALI-G	$\eta_{max} \in \{0.1, 0.01\}$	r $\in \{0, 50, 100, 200\}$
SPS	$\eta_{max} \in \{1, 10, 100\}$	$\ell_2 \in \{0, 10^{-5}, 10^{-4}, 10^{-3}\}$
$Adabound$	$\eta \in \{0.01, 0.001\}$	wd $\in \{0, 10^{-5}, 10^{-4}, 10^{-3}\}$
$Adam$	$\eta \in \{0.01, 0.001\}$	wd $\in \{0, 10^{-5}, 10^{-4}, 10^{-3}\}$
$AdamP$	$\eta \in \{0.01, 0.001\}$	wd $\in \{0, 10^{-5}, 10^{-4}, 10^{-3}\}$
$Coin$	N/A	$\ell_2 \in \{0, 10^{-5}, 10^{-4}, 10^{-3}\}$
SLS_{Armijo}	N/A	$\ell_2 \in \{0, 10^{-5}, 10^{-4}, 10^{-3}\}$
$SLS_{Goldstein}$	$\eta_{max} \in \{1, 10\}$	$\ell_2 \in \{0, 10^{-5}, 10^{-4}, 10^{-3}\}$
SLS_{Polyak}	N/A	$\ell_2 \in \{0, 10^{-5}, 10^{-4}, 10^{-3}\}$
PAL	$\eta_{max} \in \{1, 10\}$	$\ell_2 \in \{0, 10^{-5}, 10^{-4}, 10^{-3}\}$
ALI-G+	$\eta_{max} \in \{0.1, 0.01\}$	wd $\in \{0, 10^{-5}, 10^{-4}, 10^{-3}\}$

Table D.2: *Hyperparameters cross-validated in the experiments in Section 5.4.2. All other hyperparameters were left at the default values as specified by the authors' implementation (PAL, SLS, SPS, Coin, AdamP, ALI-G, Adabound) or the PyTorch implementation (SGD, Adam).*

Appendix E

Appendix: Proofs of Theorems in Chapter 7

E.1 Deviation of update

Lemma 15. *The minimiser of equation (7.1) is given by:*

$$\boldsymbol{w}_{t+1}^{b} = \Pi_{[-1,1]}\left(\frac{1}{1-2\lambda_t\eta_t}(\boldsymbol{w}_t^{b} - \eta_t\nabla\ell_{z_t}(\boldsymbol{w}_t^{b}))\right),$$
$$\boldsymbol{w}_{t+1}^{r} = \boldsymbol{w}_t^{r} - \eta_t\nabla\ell_{z_t}(\boldsymbol{w}_t).$$

Where $\Pi_{[-1,1]}$ *is element-wise projection onto* $[-1,1]$.

Proof. We first restart from equation (7.1):

$$\underset{\boldsymbol{w}\in\Omega}{\operatorname{argmin}}\big\{\frac{1}{2\eta_t}\|\boldsymbol{w}-\boldsymbol{w}_t\|^2 + \ell_{z_t}(\boldsymbol{w}_t) + \nabla\ell_{z_t}(\boldsymbol{w}_t)^\top(\boldsymbol{w}-\boldsymbol{w}_t) - \lambda_t\|\boldsymbol{w}^b\|^2\big\}.$$

where $\Omega \triangleq [-1,1]^p \cup \mathbb{R}^{d-p}$. As Ω is convex set we can solve the unconstrained

problem then projecting onto Ω.

$$\frac{\partial}{\partial \boldsymbol{w}}\left(\frac{1}{2\eta_t}\|\boldsymbol{w}-\boldsymbol{w}_t\|^2+\ell_{z_t}(\boldsymbol{w}_t)+\nabla\ell_{z_t}(\boldsymbol{w}_t)^\top(\boldsymbol{w}-\boldsymbol{w}_t)-\lambda_t\|\boldsymbol{w}^b\|^2\right), \tag{E.1}$$

$$=\frac{1}{\eta_t}(\boldsymbol{w}-\boldsymbol{w}_t)+\nabla\ell_{z_t}(\boldsymbol{w}_t)-2\lambda_t\boldsymbol{w}^b. \tag{E.2}$$

Setting (E.2) to zero and rearranging gives:

$$0=\frac{1}{\eta_t}(\boldsymbol{w}-\boldsymbol{w}_t)+\nabla\ell_{z_t}(\boldsymbol{w}_t)-2\lambda_t\boldsymbol{w}^b. \tag{E.3}$$

Next we consider $\boldsymbol{w}^b$ and $\boldsymbol{w}^r$ separately starting with $\boldsymbol{w}^r$:

$$0=\frac{1}{\eta_t}(\boldsymbol{w}^r-\boldsymbol{w}_t^r)+\nabla\ell_{z_t}(\boldsymbol{w}_t^r), \tag{E.4}$$

$$\boldsymbol{w}^r=\boldsymbol{w}_t^r-\eta_t\nabla\ell_{z_t}(\boldsymbol{w}_t). \tag{E.5}$$

Considering $\boldsymbol{w}^b$:

$$0=\frac{1}{\eta_t}(\boldsymbol{w}^b-\boldsymbol{w}_t^b)+\nabla\ell_{z_t}(\boldsymbol{w}_t^b)-2\lambda_t\boldsymbol{w}^b, \tag{E.6}$$

$$\boldsymbol{w}-2\eta_t\lambda_t\boldsymbol{w}^b=\boldsymbol{w}_t-\eta_t\nabla\ell_{z_t}(\boldsymbol{w}_t), \tag{E.7}$$

$$\boldsymbol{w}^b=\frac{1}{1-2\lambda_t\eta_t}(\boldsymbol{w}_t^b-\eta_t\nabla\ell_{z_t}(\boldsymbol{w}_t^b)). \tag{E.8}$$

Thus projecting (E.8) to Ω using $\Pi_{[-1,1]}$ gives the desired result. □

E.2 Proof of Theorem 4

Here we provide a proof for Theorem 4, this result follows the proof given in Bai et al. (2019) and is well known in the proximal algorithms literature.

Theorem 4 (BNEW). *We assume that f is β-smooth. Let $F_* \triangleq \min_\Omega F_\lambda(\boldsymbol{w})$. We further assume that $\eta_t = \frac{1}{2\beta}$, $\forall t$ and we have access to the batch gradient ∇f and*

$\lambda_t = \lambda$ then if we use BNEW with updates (6.7) and (7.3) for T steps we have:

$$\|\nabla F_\lambda(\boldsymbol{w}_{T_{best}})\|^2 \leq \frac{C\beta(F_\lambda(\boldsymbol{w}_0) - F_*)}{T}, \tag{7.4}$$

where $C > 0$ is a constant and T_{best} is defined as $T_{best} \triangleq \operatorname{argmin}_{1\leq t\leq T} \|\boldsymbol{w}_t - \boldsymbol{w}_{t-1}\|$.

Proof. At each time step t we solve the following proximal problem:

$$\boldsymbol{w}_{t+1} = \operatorname*{argmin}_{\boldsymbol{w}\in\Omega} \big\{ \frac{1}{2\eta_t}\|\boldsymbol{w} - \boldsymbol{w}_t\|^2 + f(\boldsymbol{w}_t) + \nabla f(\boldsymbol{w}_t)^\top(\boldsymbol{w} - \boldsymbol{w}_t) + \lambda R(\boldsymbol{w}) \big\}. \tag{E.9}$$

As $\boldsymbol{w}_{t+1}$ minimises the above objective we get:

$$F_\lambda(\boldsymbol{w}_t) \triangleq f(\boldsymbol{w}_t) + \lambda R(\boldsymbol{w}_t), \tag{E.10}$$

$$\geq \frac{1}{2\eta_t}\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_t\|^2 + f(\boldsymbol{w}_{t+1}) + \nabla f(\boldsymbol{w}_t)^\top(\boldsymbol{w}_{t+1} - \boldsymbol{w}_t) + \lambda R(\boldsymbol{w}_{t+1}). \tag{E.11}$$

Now using smoothness of f:

$$F_\lambda(\boldsymbol{w}_t) \geq \left(\frac{1}{2\eta_t} - \frac{\beta}{2}\right)\|\boldsymbol{w} - \boldsymbol{w}_t\|^2 + f(\boldsymbol{w}_{t+1}) + \nabla f(\boldsymbol{w}_t)^\top(\boldsymbol{w}_{t+1} - \boldsymbol{w}_t) + \lambda R(\boldsymbol{w}_{t+1}). \tag{E.12}$$

Thus, we have the following recursive relationship:

$$F_\lambda(\boldsymbol{w}_t) \geq F_\lambda(\boldsymbol{w}_{t+1}) + \frac{\beta}{2}\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_t\|^2. \tag{E.13}$$

Telescoping (E.14) for $t = 0, ..., T-1$ we get:

$$F_\lambda(\boldsymbol{w}_0) \geq F_\lambda(\boldsymbol{w}_T) + \frac{\beta}{2}\sum_{t=0}^{T-1}\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_t\|^2. \tag{E.14}$$

Rearranging:

$$\sum_{t=0}^{T-1} \|\boldsymbol{w}_{t+1} - \boldsymbol{w}_t\|^2 \leq \frac{2(F_\lambda(\boldsymbol{w}_0) - F_\lambda(\boldsymbol{w}_T))}{\beta} \leq \frac{2(F_\lambda(\boldsymbol{w}_0) - F_*)}{\beta}. \tag{E.15}$$

Therefore, we arrive at the proximity guarantee:

$$\min_{1 \leq t \leq T} \|\boldsymbol{w}_t - \boldsymbol{w}_{t-1}\| \leq \frac{2(F_\lambda(\boldsymbol{w}_0) - F_*)}{\beta T}. \tag{E.16}$$

The first-order optimality condition for $\boldsymbol{w}_{t+1}$ gives:

$$\nabla f(\boldsymbol{w}_t) + \frac{1}{\eta_t}(\boldsymbol{w} - \boldsymbol{w}_t)^2 + \nabla \lambda_t R(\boldsymbol{w}_{t+1}) = 0 \tag{E.17}$$

Combining with (E.17) and the smoothness of ℓ_z:

$$\begin{aligned} \|\nabla F_\lambda(\boldsymbol{w}_{t+1})\| &= \|\nabla f(\boldsymbol{w}_{t+1}) + \lambda R(\boldsymbol{w}_{t+1})\| && \text{(E.18)} \\ &= \|\nabla f(\boldsymbol{w}_{t+1}) - \nabla f(\boldsymbol{w}_t) - \frac{1}{\eta_t}(\boldsymbol{w} - \boldsymbol{w}_t)^2\| && \text{(E.19)} \\ &\leq \left(\frac{1}{\eta} + \beta\right) \|\boldsymbol{w} - \boldsymbol{w}_t\| = 3\beta \|\boldsymbol{w} - \boldsymbol{w}_t\| && \text{(E.20)} \end{aligned}$$

Inserting $t = T_{best} - 1$ and applying (E.16), we obtain the desired result.

$$\|\nabla F_\lambda(\boldsymbol{w}_{T_{best}})\|^2 \leq 9\beta^2 \|\boldsymbol{w}_{T_{best}} - \boldsymbol{w}_{T_{best}-1}\|^2, \tag{E.21}$$

$$\|\nabla F_\lambda(\boldsymbol{w}_{T_{best}})\|^2 \leq 9\beta^2 \operatorname*{argmin}_{1 \leq t \leq T} \|\boldsymbol{w}_t - \boldsymbol{w}_{t-1}\|^2 \leq \frac{18\beta(F_\lambda(\boldsymbol{w}_0) - F_*)}{T}. \tag{E.22}$$

□

References

Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Man, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Vi, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org.

Ajanthan, T., K. Gupta, P. Torr, R. Hartley, and P. Dokania (2021). Mirror descent view for neural network quantization. *International Conference on Artificial Intelligence and Statistics*.

Armijo, L. (1966). Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics*.

Asi, H. and J. C. Duchi (2019). Stochastic (approximate) proximal point methods: Convergence, optimality, and adaptivity. *SIAM Journal on Optimization*.

Auslender, A. (2009). Bundle methods for machine learning. *Numerical Methods for Nondifferentiable Convex Optimization. Mathematical Programming Study*.

Bach, F. R., R. Jenatton, J. Mairal, and G. Obozinski (2011). Optimization with sparsity-inducing penalties. *CoRR*.

Bai, Y., Y.-X. Wang, and E. Liberty (2019). Proxquant: Quantized neural networks via proximal operators. *International Conference on Learning Representations*.

Bernstein, J., Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar (2018). signsgd: Compressed optimisation for non-convex problems. *International Conference on Machine Learning*.

Berrada, L., A. Zisserman, and M. P. Kumar (2019). Deep Frank-Wolfe for neural network optimization. *International Conference on Learning Representations*.

Berrada, L., A. Zisserman, and M. P. Kumar (2020). Training neural networks for and by interpolation. *International Conference on Machine Learning*.

Berrada, L., A. Zisserman, and M. P. Kumar (2021). Comment on stochastic polyak step-size: Performance of ali-g. *arXiv preprint arXiv:2105.10011*.

Bertsekas, D. P. (2009). *Convex Optimization Theory*. Athena Scientific.

Bethge, J., H. Yang, M. Bornstein, and C. Meinel (2019). Back to simplicity: How to train accurate bnns from scratch? *arXiv preprint arXiv:1906.08637*.

Bommasani, R., D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, et al. (2021). On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

Bowman, S. R., G. Angeli, C. Potts, and C. D. Manning (2015). A large annotated corpus for learning natural language inference. *Conference on Empirical Methods in Natural Language Processing*.

Brown, T. B., B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. (2020). Language models are few-shot learners. *Neural Information Processing Systems*.

Bubeck, S. (2015). Convex optimization: Algorithms and complexity. *Foundations and Trends in Machine Learning*.

Bubeck, S. and M. Sellke (2021). A universal law of robustness via isoperimetry. *Neural Information Processing Systems*.

Chang, C.-C. and C.-J. Lin (2011). Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*.

Chen, J. and Q. Gu (2018). Padam: Closing the generalization gap of adaptive gradient methods in training deep neural networks. *arXiv preprint*.

Chen, X., S. Liu, R. Sun, and M. Hong (2019). On the convergence of a class of adam-type algorithms for non-convex optimization. *International Conference on Learning Representations*.

Conneau, A., D. Kiela, H. Schwenk, L. Barrault, and A. Bordes (2017). Supervised learning of universal sentence representations from natural language inference data. *Conference on Empirical Methods in Natural Language Processing*.

Courbariaux, M., Y. Bengio, and J.-P. David (2015). Binaryconnect: Training deep neural networks with binary weights during propagations. *Neural Information Processing Systems*.

Davtyan, A., S. Sameni, L. Cerkezi, G. Meishvili, A. Bielski, and P. Favaro (2022). Koala: A kalman optimization algorithm with loss adaptivity. *Proceedings of the AAAI Conference on Artificial Intelligence*.

Défossez, A. and F. Bach (2017). Adabatch: Efficient gradient aggregation rules for sequential and parallel stochastic gradient methods. *arXiv preprint*.

Défossez, A., L. Bottou, F. Bach, and N. Usunier (2020). A simple convergence proof of adam and adagrad.

Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei (2009). Imagenet: A large-scale hierarchical image database. *Conference on Computer Vision and Pattern Recognition*.

Deng, L., P. Jiao, J. Pei, Z. Wu, and G. Li (2018). Gxnor-net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework. *Conference on Computer Vision and Pattern Recognition*.

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. *Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

Du, M., S. Mukherjee, Y. Cheng, M. Shokouhi, X. Hu, and A. H. Awadallah (2021). What do compressed large language models forget? robustness challenges in model compression. *arXiv preprint arXiv:2110.08419*.

Duchi, J., E. Hazan, and Y. Singer (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*.

Frank, M. and P. Wolfe (1956). An algorithm for quadratic programming. *Naval Research Logistics Quarterly*.

Galanti, T., A. György, and M. Hutter (2021). On the role of neural collapse in transfer learning. *International Conference on Learning Representations*.

Gholami, A., S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer (2021). A survey of quantization methods for efficient neural network inference. *Conference on Computer Vision and Pattern Recognition*.

Goodfellow, I. J., Y. Bengio, and A. Courville (2016). *Deep Learning*. MIT Press.

Gower, R. M., A. Defazio, and M. Rabbat (2021). Stochastic polyak stepsize with a moving target. *OPT Annual Workshop on Optimization for Machine Learning*.

Hao, Z., Y. Jiang, H. Yu, and H.-D. Chiang (2021). Adaptive learning rate and momentum for training deep neural networks.

Hazan, E. and S. M. Kakade (2022). Revisiting the polyak step size. *https://arxiv.org/pdf/1905.00313.pdf*.

He, K., X. Zhang, S. Ren, and J. Sun (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *International Conference on Computer Vision*.

He, K., X. Zhang, S. Ren, and J. Sun (2016). Deep residual learning for image recognition. *Conference on Computer Vision and Pattern Recognition*.

Helwegen, K., J. Widdicombe, L. Geiger, Z. Liu, K.-T. Cheng, and R. Nusselder (2019). Latent weights do not exist: Rethinking binarized neural network optimization. *Neural Information Processing Systems*.

Heo, B., S. Chun, S. J. Oh, D. Han, S. Yun, G. Kim, Y. Uh, and J.-W. Ha (2021). Adamp: Slowing down the slowdown for momentum optimizers on scale-invariant weights. *International Conference on Learning Representations*.

Hornik, K., M. Stinchcombe, and H. White (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*.

Howard, A. G., M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam (2019). Mobilenets: Efficient convolutional neural networks for mobile vision applications.

Huang, G., Z. Liu, K. Q. Weinberger, and L. van der Maaten (2017). Densely connected convolutional networks. *Conference on Computer Vision and Pattern Recognition*.

Huang, X., M. Kwiatkowska, S. Wang, and M. Wu (2017). Safety verification of deep neural networks. *International Conference on Computer Aided Verification*.

Hubara, I., M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio (2016). Binarized neural networks. *Neural Information Processing Systems*.

Ioffe, S. and C. Szegedy (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International Conference on Machine Learning*.

Kingma, D. P. and J. Ba (2015). Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Kingma, D. P. and M. Welling (2014). Auto-encoding variational Bayes. *International Conference on Learning Representations*.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. *Technical Report*.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*.

Lapin, M., M. Hein, and B. Schiele (2016). Loss functions for top-k error: Analysis and insights. *Conference on Computer Vision and Pattern Recognition*.

LeCun, Y., Y. Bengio, and G. Hinton (2015). Deep learning. *Nature*.

Lemaréchal, C., A. Nemirovski, and Y. Nesterov (1995). New variants of bundle methods. *Math. Program*.

Levy, K. (2017). Online to offline conversions, universality and adaptive minibatch sizes. *Neural Information Processing Systems*.

Li, F., B. Zhang, and B. Liu (2016). Ternary weight networks. *Conference on Computer Vision and Pattern Recognition*.

Li, M., M. Soltanolkotabi, and S. Oymak (2020). Gradient descent with early stopping is provably robust to label noise for overparameterized neural networks. *Journal of Machine Learning Research*.

Lin, X., C. Zhao, and W. Pan (2017). Towards accurate binary convolutional neural network. *Neural Information Processing Systems*.

Liu, C., T. Arnon, C. Lazarus, C. Barrett, and M. J. Kochenderfer (2019). Algorithms for verifying deep neural networks. *arXiv:1903.06758*.

Liu, Z., Z. Shen, M. Savvides, and K.-T. Cheng (2020). Reactnet: Towards precise binary neural network with generalized activation functions. *European Conference on Computer Vision*.

Liu, Z., B. Wu, W. Luo, X. Yang, W. Liu, and K.-T. Cheng (2018a). Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. *European Conference on Computer Vision*.

Liu, Z., B. Wu, W. Luo, X. Yang, W. Liu, and K.-T. Cheng (2018b). Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. *European Conference on Computer Vision*.

Loizou, N., S. Vaswani, I. Laradji, and S. Lacoste-Julien (2021). Stochastic polyak step-size for sgd: An adaptive learning rate for fast convergence. *International Conference on Artificial Intelligence and Statistics*.

Loshchilov, I. and F. Hutter (2017). SGDR: Stochastic gradient descent with warm restarts. *International Conference on Learning Representations*.

Loshchilov, I. and F. Hutter (2019). Fixing weight decay regularization in adam. *International Conference on Learning Representations*.

Luo, L., Y. Xiong, Y. Liu, and X. Sun (2019). Adaptive gradient methods with dynamic bound of learning rate. *International Conference on Learning Representations*.

Ma, S., R. Bassily, and M. Belkin (2018). The power of interpolation: Understanding the effectiveness of sgd in modern over-parametrized learning. *International Conference on Machine Learning*.

Ma, X., B. Li, Y. Wang, S. M. Erfani, S. Wijewickrema, G. Schoenebeck, D. Song, M. E. Houle, and J. Bailey (2018). Characterizing adversarial subspaces using local intrinsic dimensionality. *International Conference on Learning Representations*.

Martinez, B., J. Yang, A. Bulat, and G. Tzimiropoulos (2020). Training binary neural networks with real-to-binary convolutions. *International Conference on Learning Representations*.

Mukkamala, M. C. and M. Hein (2017). Variants of rmsprop and adagrad with logarithmic regret bounds. *International Conference on Machine Learning*.

Mutschler, M. and A. Zell (2020). Parabolic approximation line search for dnns. *Neural Information Processing Systems*.

Neill, J. O. (2020). An overview of neural network compression. *arXiv preprint arXiv:2006.03669*.

Nemirovsky, A., D. Yudin, and E. Dawson (1983). *Problem Complexity and Method Efficiency in Optimization*.

Nesterov, Y. (1983). A method of solving a convex programming problem with convergence rate $\mathcal{O}(1/k^2)$. *Soviet Mathematics Doklady*.

Netzer, Y., T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng (2011). Reading digits in natural images with unsupervised feature learning. *Neural Information Processing Systems*.

Oberman, A. M. and M. Prazeres (2019). Stochastic gradient descent with polyak's learning rate. *arXiv preprint*.

Orabona, F. and D. Pál (2015). Scale-free algorithms for online linear optimization. *International Conference on Algorithmic Learning Theory*.

Paszke, A., S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer (2017). Automatic differentiation in pytorch. *NIPS Autodiff Workshop*.

Polyak, B. T. (1969). Minimization of unsmooth functionals. *USSR Computational Mathematics and Mathematical Physics*.

Ragab, D. A., M. A. Sharkas, S. Marshall, and J. Ren (2019). Breast cancer detection using deep convolutional neural networks and support vector machines. *PeerJ*.

Ramesh, A., P. Dhariwal, A. Nichol, C. Chu, and M. Chen (2022). Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*.

Rastegari, M., V. Ordonez, J. Redmon, and A. Farhadi (2016). Xnor-net: Imagenet classification using binary convolutional neural networks. *European Conference on Computer Vision*.

Reddi, S. J., S. Kale, and S. Kumar (2018). On the convergence of adam and beyond. *International Conference on Learning Representations*.

Robbins, H. and S. Monro (1951). A stochastic approximation method. *The annals of mathematical statistics*.

Rolinek, M. and G. Martius (2018). L4: Practical loss-based stepsize adaptation for deep learning. *Neural Information Processing Systems*.

Saharia, C., W. Chan, S. Saxena, L. Li, J. Whang, E. Denton, S. K. S. Ghasemipour, B. K. Ayan, S. S. Mahdavi, R. G. Lopes, et al. (2022). Photorealistic text-to-image diffusion models with deep language understanding. *arXiv preprint arXiv:2205.11487*.

Schneider, F., L. Balles, and P. Hennig (2019). DeepOBS: A deep learning optimizer benchmark suite. *International Conference on Learning Representations*.

Shazeer, N. and M. Stern (2018). Adafactor: Adaptive learning rates with sublinear memory cost. *International Conference on Machine Learning*.

Shoeybi, M., M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro (2019). Megatron-lm: Training multi-billion parameter language models using gpu model parallelism. *arXiv preprint*.

Sivaprasad, P. T., F. Mai, T. Vogels, M. Jaggi, and F. Fleuret (2020). Optimizer benchmarking needs to account for hyperparameter tuning. *International Conference on Machine Learning*.

Smola, A. J., S. V. N. Vishwanathan, and Q. V. Le (2007). Bundle methods for machine learning. *Neural Information Processing Systems*.

Suarez-Ramirez, C. D., M. Gonzalez-Mendoza, L. Chang, G. Ochoa-Ruiz, and M. A. Duran-Vega (2021). A bop and beyond: a second order optimizer for binarized neural networks. *LatinX in CV Research Workshop at CVPR*.

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich (2015). Going deeper with convolutions. *Conference on Computer Vision and Pattern Recognition*.

Tieleman, T. and G. Hinton (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*.

Van Den Oord, A., S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu (2016). Wavenet: A generative model for raw audio. *SSW 125*, 2.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin (2017). Attention is all you need. *Neural Information Processing Systems*.

Vaswani, S., F. Bach, and M. Schmidt (2019). Fast and faster convergence of sgd for over-parameterized models and an accelerated perceptron. *International Conference on Artificial Intelligence and Statistics*.

Vaswani, S., A. Mishkin, I. Laradji, M. Schmidt, G. Gidel, and S. Lacoste-Julien (2019). Painless stochastic gradient: Interpolation, line-search, and convergence rates. *Neural Information Processing Systems*.

Wan, D., F. Shen, L. Liu, F. Zhu, J. Qin, L. Shao, and H. T. Shen (2018). Tbn: Convolutional neural network with ternary inputs and binary weights. *European Conference on Computer Vision*.

Wang, N., J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan (2018). Training deep neural networks with 8-bit floating point numbers. *Neural Information Processing Systems*.

Wilson, A. C., R. Roelofs, M. Stern, N. Srebro, and B. Recht (2017). The marginal value of adaptive gradient methods in machine learning. *Neural Information Processing Systems*.

Yuan, L., D. Chen, Y.-L. Chen, N. Codella, X. Dai, J. Gao, H. Hu, X. Huang, B. Li, C. Li, et al. (2021). Florence: A new foundation model for computer vision. *arXiv preprint arXiv:2111.11432*.

Zagoruyko, S. and N. Komodakis (2016). Wide residual networks. *British Machine Vision Conference*.

Zaheer, M., S. Reddi, D. Sachan, S. Kale, and S. Kumar (2018). Adaptive methods for nonconvex optimization. *Neural Information Processing Systems*.

Zeiler, M. ADADELTA: an adaptive learning rate method. *arXiv preprint*.

Zheng, S. and J. T. Kwok (2017). Follow the moving leader in deep learning. *International Conference on Machine Learning*.

Zhou, S., Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou (2016). Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*.

Zhou, Y., J. Yang, H. Zhang, Y. Liang, and V. Tarokh (2019). Sgd converges to global minimum in deep learning via star-convex path. *International Conference on Learning Representations*.

Zhu, C., S. Han, H. Mao, and W. J. Dally (2017). Trained ternary quantization. *International Conference on Learning Representations*.