Leveraging Structure for Optimization in Deep Learning

Leonard Berrada

Wadham College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Michaelmas 2019

Acknowledgements

First and foremost, I would like to thank my supervisors, Andrew and Pawan[1], from whom I have learned so much. I am indebted to their patience throughout my doctorate. More specifically, I thank Andrew for encouraging me to ground my research in practical applications, and for teaching me the importance of clarity and scientific curiosity. I can only hope to ever understand a subset with non-zero measure of his scientific knowledge. I have the utmost gratitude to Pawan for teaching me how to disentangle intricate problems to their simplest form, for communicating his passion for optimization and problem-solving, and all in all, for always providing exactly on-point advice. I could not have wished for a better daily advisor.

My sincere thanks also go to my fellow labmates over the years: Diane, Pankaj, Rudy, Alban, Prateek, Jodie, Florian, Alasdair and Alessandro. I really have learned a lot from such amazing colleagues. In particular, thanks to Rudy for all the discussions about optimization, writing and reviewing papers, as well as more important topics such as whisky, baking and why rowing is the best thing in life – almost surely. Thanks also to Alban for all the help and discussions about coding and PyTorch – and also for an amazing demonstration at splitting logs of wood after NeurIPS 2016.

I would also like to thank Pierre and Freddy for bearing with my usually short, often objective and always complaint-free tales (not rants) about research and publishing in machine learning; Camille, for hosting me so many times in London; and of course Clément, Guillaume, Victoire, Dac-Long and Michélène, for remaining amazing and supportive friends.

I also thank my Oxford friends, who have made my experience in Oxford so much more enjoyable. Thanks to the 450 crew: KP, Nikitas, Adam, Louise and Kim; to Bernie (who made me learn a lot of Spanish against my will), Gabriele, Ada, Penny, it has been great fun. Of course, my very special thanks go to Paula, for her support in so many ways that they cannot be put into words here.

I would also like to particularly thank my family for having guided, advised and supported me throughout my entire education. They deserve and have most credits for making a doctorate possible for me.

Finally, I am very grateful to YouGov and the EPSRC for funding my doctorate.

[1]Advisors sorted by alphabetical order here.

Abstract

In the past decade, neural networks have demonstrated impressive performance in supervised learning. They now power many applications ranging from real-time medical diagnosis to human-sounding virtual assistants through wild animal monitoring. Despite their increasing importance however, they remain difficult to train due to a complex interplay between the learning objective, the optimization algorithm and generalization performance. Indeed, using different loss functions and optimization algorithms lead to trained models with significantly different performances on unseen data.

In this thesis, we focus first on the loss function, for which using a task-specific approach can improve the generalization performance in the small or noisy data setting. Specifically, we consider the top-k classification setting. We show that traditional piecewise-linear top-k loss functions require smoothing to work well with neural networks. However, it is computationally challenging to evaluate the resulting smoothed loss function and its gradient. Indeed, using a naive approach would result in a runtime proportional to the number of combinations of possible k-predictions. Thanks to a connection to polynomial algebra, we develop computationally efficient algorithms to evaluate the smoothed loss function and its gradient. This allows us to train models with stochastic gradient descent (SGD) using the smooth top-k loss function. We show that doing so is more robust to over-fitting than using the standard cross-entropy loss function.

Second, we turn our attention to optimization algorithms. Indeed, while SGD empirically provides good generalization, it requires a manually tuned learning-rate schedule. Obtaining a suitable learning-rate schedule for a given network and data set is a time-consuming and computationally expensive task. In this thesis, we propose novel optimization algorithms to alleviate this issue. In particular, we propose to exploit the structure in three different ways – each one leading to a new optimization algorithm. First, we exploit the piecewise linearity of the activation and loss functions, which results in a difference-of-convex programming approach. Second, we use the compositionality of the model and the loss function with the help of a proximal approach. Third, we exploit the property of interpolating models to derive an adaptive learning-rate for SGD. Empirically, we compare the performance of the three algorithms on various deep learning tasks, and we demonstrate their advantages over state-of-the-art methods while avoiding the need for manual learning rate schedules.

Contents

List of Figures

Truth is ever to be found in simplicity, and not in the multiplicity and confusion of things.

— Sir Isaac Newton

1 Introduction

Contents

1.1 Preamble

Deep neural networks have proven to be extremely successful for supervised learning in the past decade (LeCun et al., 2015). Beyond key breakthroughs on academic challenges (Krizhevsky et al., 2012; Devlin et al., 2019), they are now powering applications in our everyday life, such as human-sounding virtual assistants (Oord et al., 2016), monitoring animals in the wild (Norouzzadeh et al., 2018) or providing early medical diagnosis (Tomavsev et al., 2019). As research in machine perception makes further progress, one can only expect to see the range of applications grow in the next few years.

However, the difficulty to train neural networks is arguably an impediment to the rate of this progress. Indeed, traditional loss functions, which have most often been originally designed for linear models, can cause optimization issues when combined with deep neural networks. Furthermore, training neural networks still involves a significant amount of manual tuning, in particular for the optimizer learning-rate.

In this thesis, we leverage the structure of various problems to provide solutions to the aforementioned issues. We also draw inspiration from support vector machines, and more generally from the convex optimization literature, to adapt long-established ideas for the deep learning setting.

1.2 Optimization and Deep Learning

As per the predominant approach in deep learning, training a neural network can be formulated as an optimization problem. In more details, such a task can be sub-divided into two aspects: how to formulate the learning objective, and how to minimize it. Both of these aspects affect the ability of models to generalize on unseen data, and can benefit from solutions tailored to the deep learning setting.

The choice of learning objective can have an important effect on generalization. In particular, task-agnostic loss functions, such as the commonly used cross-entropy loss, are prone to overfitting when little training data is available. This effect is

exacerbated by the over-parameterization of modern neural-networks, and it can result in poor generalization performance in practice.

The minimization procedure also impacts generalization performance. Indeed, since there are potentially many stationary points for neural networks, various choices of algorithms may lead to widely different solutions, each with their own generalization performance. More specifically, it has been observed that stochastic gradient descent (SGD) tends to find solutions with significantly better generalization performance than adaptive gradient methods such as Adagrad (Duchi et al., 2011) or Adam (Kingma and Ba, 2015). However, SGD requires extensive tuning from the user, which is a time-consuming and computationally expensive task. In contrast, easier to tune alternatives, such as adaptive gradient methods, tend to provide sub-par generalization performance.

Overall, this calls for designing (i) task-specific loss functions that perform well with deep neural networks, and (ii) optimization methods that exploit SGD's empirical strengths, while removing its need for manual tuning. These two points will be the topic of investigation of this thesis. In order to give context to this work, we begin with a more detailed description of the problem. This will shed light on existing solutions as well as on their shortcomings.

1.3 Formulating the Learning Objective

The learning objective defines the function whose minimization leads to a trained model. Note that in this thesis, we use the following terminology: the learning objective is the objective function of the optimization problem; the loss function is the function defined per sample that encourages learning; the regularization is the function that limits the capacity of the model in order to promote generalization. In other words, the learning objective is the sum of the regularization and the loss function.

The regularization usually limits the expression capacity of the model, which in turn encourages its generalization performance (Friedman et al., 2001; Goodfellow et al., 2016). However, understanding regularization in deep learning is challenging (Neyshabur et al., 2017; Zhang et al., 2017a) and the design of novel regularizing

methods is an active area of research (Zhang et al., 2018; Bietti et al., 2019). Empirically, employing regularization does not suffice to yield good generalization if the loss function is prone to over-fitting, thus the choice of loss function remains essential.

Overall, the loss function has to satisfy two requirements to be practically useful. The first requirement is that its minimization is computationally feasible, or, in other words, that it is "easy to optimize". In the past era of linear models, "easy to optimize" usually meant that the loss function was convex. Indeed, when composing it with a linear model (and a convex regularization), this would lead to a convex optimization problem, which could then be solved with a standard convex solver. In deep learning, convexity of the loss function has lost its importance: even with a convex loss function, the resulting optimization problem is non-convex for (non-linear) neural networks. It currently remains an open question what it takes precisely for a loss function to be "easy to optimize" with modern neural networks. This thesis brings some elements of answers through empirical observations in the top-k classification setting.

Assuming tractability of the problem, the second requirement of the loss function is that it is representative of the learning task. In other words, accurate minimization of the learning objective should lead to a model that makes correct predictions on the training set, as well as on unseen data. There are two main approaches to design the loss function for this: (i) encouraging the model to mimic the empirical data distribution or (ii) penalizing incorrect predictions in a task-specific way. Both of these approaches are discussed below.

1.3.1 Maximum Likelihood

The maximum-likelihood approach formulates the learning objective as the negative log-likelihood of the data. Minimizing such a learning objective is then equivalent to maximizing the likelihood of the data. Equivalently, the learning objective measures a "distance" (KL-divergence) between the estimated distribution and the empirical distribution. For standard classification, this results in the commonly used cross-entropy loss.

The advantage of this framework is that it is both powerful and task-agnostic, which makes it widely applicable and easy to use. However, its downside is that encouraging the model to perfectly match the empirical distribution may lead to overfitting, in particular when little data is available or when the labels contain noise.

1.3.2 Empirical Risk Minimization

In contrast with the task-agnostic approach of maximum-likelihood, it is possible, given a model prediction and a ground truth label, to define a penalty in a task-specific way. Such a penalty will be referred to as the prediction loss.

As a motivating example, consider a hypothetical data set of images, where each image contains five objects but only one of them (selected at random) is labeled as the ground truth. It is then legitimate to aim for a model that achieves a low top-5 error-rate without assigning any importance to top-1 error-rate. However, in such a case, using a maximum-likelihood approach would penalize a model making incorrect top-1 predictions, even when its top-5 predictions are correct. This can harm learning of the model, especially in the small data setting. In contrast, by using a prediction loss that cares only about top-5 predictions, we can facilitate learning and generalization of the model.

There are many other learning tasks where it is beneficial to use a task-specific prediction loss, such as object localization (Blaschko and Lampert, 2008), semantic segmentation (Everingham et al., 2010), or hierarchical classification (Cai and Hofmann, 2004; Binder et al., 2012).

In many cases however – including the aforementioned ones, the prediction loss is a discontinuous function of the model parameters: for instance, prediction of the top-5 labels is a discontinuous function of the model parameters. This discontinuity makes direct minimization difficult (Hazan et al., 2010), in particular for neural networks (Song et al., 2016).

To alleviate this issue, one can design the loss function as a continuous surrogate of this (discontinuous) prediction loss, while ensuring that its minimization leads to approximate optimization of the prediction loss. For instance, structured prediction

provides a framework to do so, by creating a continuous piecewise linear upper bound on the original loss metric (Tsochantaridis et al., 2004). Once adapted to the deep learning setting, this approach can yield significant benefits over maximum likelihood, as we demonstrate for the top-k setting in chapter 2.

Recap. Designing the right loss function is crucial for the performance of deep neural networks, especially given the fact that they are often over-parameterized. It can thus be beneficial to use task-specific loss functions, in particular in cases where the amount of available data is limited.

1.4 Minimizing the Learning Objective

We now turn to the optimization algorithm, that is, the procedure that finds the set of model parameters that minimize the learning objective.

Before describing the most commonly used optimization algorithms in deep learning, it is worth characterizing the optimization problem. As mentioned previously, the learning objective is usually a loss function averaged over training samples – potentially with a regularization term. As a function of the model parameters, the loss function is usually non-convex and non-smooth. Crucially, this objective function is usually defined over a large parameter space (the model can have up to billions of parameters (Shoeybi et al., 2019)) and over a large number of samples (also up to billions of samples (Mahajan et al., 2018)). As a consequence, it is essential for deep learning optimization algorithms to scale well with both the parameter space dimension and the number of samples.

We now describe the two most popular optimization algorithms for deep learning: stochastic gradient descent and adaptive gradient methods. This description aims to present the most commonly used methods and discuss their limitations, rather than to provide an overview of existing methods – for this, we refer the reader to the related work section of each chapter.

1.4.1 Stochastic Gradient Descent

Stochastic gradient descent (SGD) satisfies the aforementioned scalability requirements. Indeed, its computational cost per iteration scales linearly with the dimension and is independent of the number of training samples. In addition, its convergence guarantees are usually independent of the dimension. In practice, SGD also benefits from its simplicity to implement, in particular within deep learning frameworks that have automatic differentiation built in (Abadi et al., 2015; Paszke et al., 2017).

The downside of SGD, however, is that it requires a hand-designed learning-rate schedule for good performance in deep learning. It is worth noting that in convex optimization, as long as they satisfy appropriate conditions (Bertsekas, 2003), several learning-rate schedules lead to equally optimal solutions. In contrast, when using different learning-rate schedules in deep learning, SGD converges to solutions that are potentially widely different. As a result, practitioners design learning-rate schedules on a case-by-case basis: for instance, Simonyan and Zisserman (2015) and He et al. (2016) adapt the learning rate according to the validation accuracy; Huang et al. (2017) use a piecewise constant schedule; Szegedy et al. (2015) a geometrically decaying one; and Loshchilov and Hutter (2017) a cyclic one with a cosine annealing.

The need for such manual tuning incurs significant costs of human time and computational resources. This overall calls for more automatic approaches. It is also worth noting that black-box approaches that automatically tune the learning-rate are extremely computational expensive (Jaderberg et al., 2017).

1.4.2 Adaptive Gradient Methods

Adaptive gradient methods, such as the popular Adagrad (Duchi et al., 2011) or Adam (Kingma and Ba, 2015), exploit an accumulated history of past gradients to re-scale the current gradient – usually per coordinate. This has the advantage of being invariant to a loss function rescaling, and pre-conditioning gradients may help with badly conditioned problems.

As a result, these algorithms are usually easier to tune and do not to require learning-rate schedules – at least on simple tasks. However, the downside of

adaptive gradient methods is that in practice, they tend to generalize poorly for supervised learning (Wilson et al., 2017) – an observation corroborated by the experiments of this work.

Recap. SGD offers good generalization in practice but its learning-rate schedule is difficult to tune. In contrast, adaptive gradient methods are easier to tune but provide poor generalization.

1.5 Thesis Outline & Contributions

As mentioned previously, this thesis investigates the two core components of optimization for deep learning: the loss function ("how to formulate the learning objective") and the optimization algorithm ("how to minimize the learning objective"). Chapter 2 of this thesis investigates the design of loss function, while chapters 3, 4 and 5 propose novel optimization algorithms.

In what follows, we detail the content and contribution of each chapter. We point out that this thesis follows the integrated thesis format, and as such is a collection of papers. Each of the following chapters corresponds to a publication of which I am the only first author.

1.5.1 Smooth Loss Functions for Deep Top-k Classification

Corresponding publication. (Berrada et al., 2018), published at ICLR 2018.

Summary. We design a novel loss-function for top-k classification with deep neural networks. We show how to smooth the function to facilitate optimization, and we demonstrate that this approach outperforms cross-entropy on data sets that either have a small number of training samples or contain label confusion.

General Contributions. While existing top-k hinge loss functions have been shown to work well with linear models, we empirically find that they fail in combination with deep neural networks. We identify that this issue is caused by gradient sparsity, and that it can be alleviated with smoothing – an insight

that goes beyond the specific use-case of top-k classification. We show that this smoothed top-k loss function (i) can be computed efficiently, and (ii) is more robust to overfitting than cross-entropy on noisy CIFAR-100 and on ImageNet subsets.

Algorithmic Contributions. Smoothing introduces a computational challenge, as a naive approach would require $\mathcal{O}(\binom{n}{k})$ operations, with n the number of classes, and k the number of predictions desired for top-k classification. By exploiting the structure of the problem, we reduce this computational problem to expanding a polynomial with n roots up to degree k, which can be done in $\mathcal{O}(kn)$ time with existing algorithms. In order to exploit available GPU hardware, we design a new divide-and-conquer algorithm, which offers the same worst-case complexity of $\mathcal{O}(kn)$, and in practice presents a scaling of $\mathcal{O}(k \log(n))$ thanks to parallelism. Furthermore, in order to make the algorithm work in single-precision floating-point, we present an approach operating in log-space that is orders of magnitude more numerically stable. Because of the use of many operations in log-space, automatic differentiation incurs a significant slowdown. Therefore, we use a custom implementation of the gradient, for which we also derive novel heuristics to reach numerical stability in single-precision floating-point.

Theoretical Contributions. We prove that the smooth loss function converges point-wise to its non-smooth counterpart when the smoothing temperature goes to zero. We further prove that the smooth loss function is an upper-bound on its non-smooth counter part if and only if $k = 1$. Finally, we prove that the smooth loss function is, up to a scaling factor, an upper-bound on the top-k prediction loss.

1.5.2 Trusting SVMs for Piecewise Linear CNNs

Corresponding publication. (Berrada et al., 2017), published at ICLR 2017.

Summary. We design a custom optimization algorithm for learning piecewise linear Convolutional Neural Networks (CNNs), that is, CNNs for which both the activations and the loss function are continuous piecewise linear functions. This includes many common activation functions such as ReLUs, max-pooling and max-out, as well as hinge-type loss functions.

General Contributions. We design a novel layer-wise optimization algorithm that exploits the structure of the problem. Indeed, we show that optimizing one layer at a time is equivalent to a latent Support Vector Machines (SVM) problem. This has a difference-of-convex structure, for which we design an algorithm to separate explicitly the convex part from the concave one, at a computational cost comparable to that of a forward pass. As a result, we can use the Concave-Convex Procedure, which iteratively creates approximate convex problems. We show that each convex problem is a standard SVM, which can be solved efficiently with the Block-Coordinate Frank-Wolfe (BCFW) algorithm. We provide empirical evidence that our algorithm (i) scales to ImageNet, and (ii) improves the performance of state-of-the-art adaptive gradient methods on MNIST, CIFAR-10 and CIFAR-100.

Algorithmic Contributions. We customize and improve the approach to this problem in three ways. First, by employing a proximal term in the objective, we allow an approximate warm start of each convex optimization problem, which significantly speeds up convergence. Second, for the layer-wise training of fully connected layers, we identify a low-rank factorization of the matrices stored by BCFW, which results in memory savings by an order of magnitude on ImageNet. Third, we design a heuristic to restrict the dual search-space of BCFW, which yields significant speed-ups in practice.

1.5.3 Deep Frank-Wolfe For Neural Network Optimization

Corresponding publication. (Berrada et al., 2019a), published at ICLR 2019.

Summary. We design an optimization algorithm for learning any neural network with an SVM loss function (or more generally, a convex piecewise linear loss function), while requiring only one hyper-parameter for the learning-rate.

General Contributions. We exploit the compositional structure between the model and the loss function to use a proximal framework: at each iteration, we create a proximal problem where the model is linearized while the loss function is kept intact. When the loss function is an SVM, the resulting problem is exactly a linear SVM, which can thus be solved with standard SVM solvers. In practice, we use the Frank-Wolfe algorithm, which has the advantage of providing line-search in closed-form. The resulting algorithm is termed Deep Frank-Wolfe (DFW). We evaluate DFW on a bi-directional LSTM on SNLI, as well as on variants of deep residual networks on CIFAR-10 and CIFAR-100. On these tasks, we show that DFW significantly outperforms adaptive gradient methods, and is even competitive with SGD employing a manually tuned schedule.

Algorithmic Contribution. In order to avoid sparsity of the gradients, which causes difficulty for optimization (chapter 2), we introduce a novel and hyper-parameter free dual smoothing of the loss function. This results in non-sparse gradients in the primal. We show that we can automatically detect when this does not result in an ascent direction, in which case the algorithm falls back to the default direction. This approach considerably helps optimization in practice.

1.5.4 Training Neural Networks for and by Interpolation

Corresponding publication. (Berrada et al., 2019b), under review.

Summary. We design an optimization algorithm for any deep learning task where the model is able to reach a loss function value of near zero on all training samples simultaneously (assuming the loss function is non-negative).

General Contributions. We present an adaptive learning-rate for the interpolation setting, which yields an algorithm with the same computational cost per iteration as SGD. We empirically find that this learning-rate tends to over-shoot in the non-convex setting, and therefore introduce a maximal learning-rate hyper-parameter. Using a maximal learning-rate has a natural proximal interpretation as an extension of SGD that takes into account a lower-bound of the loss function. This results in an algorithm termed Adaptive Learning-rates for Interpolation with Gradients (ALI-G). Since the hyper-parameter is kept constant, ALI-G requires only one hyper-parameter for the learning-rate. We connect the ALI-G algorithm to many existing methods, including the Polyak step-size, L4, aProx and DFW. We assess the performance of ALI-G on differentiable computers, a bi-directional LSTM on SNLI, as well as on variants of deep residual networks on SVHN, CIFAR-10 and CIFAR-100.

Theoretical Contributions. For stochastic optimization, we prove the convergence rates of ALI-G in the following settings: $\mathcal{O}(1/\sqrt{T})$ for convex Lipschitz functions, $\mathcal{O}(1/T)$ for convex smooth functions and $\mathcal{O}(\exp(-\alpha T/8\beta)$ for α-strongly convex and β-smooth functions. In addition, we prove similar results when employing a constant maximal learning-rate: the convergence rate is identical for a sufficiently large maximal learning-rate, and is slowed down by the maximal learning-rate value if it is too low.

2

Smooth Loss Functions for Deep Top-k Classification

Contents

Abstract

The top-k error is a common measure of performance in machine learning and computer vision. In practice, top-k classification is typically performed with deep neural networks trained with the cross-entropy loss. Theoretical results indeed suggest that cross-entropy is an optimal learning objective for such a task in the limit of infinite data. In the context of limited and noisy data however, the use of a loss function that is specifically designed for top-k classification can bring significant improvements. Our empirical evidence suggests that the loss function must be smooth and have non-sparse gradients in order to work well with deep neural networks. Consequently, we introduce a family of smoothed loss functions that are suited to top-k optimization via deep learning. The widely used cross-entropy is a special case of our family. Evaluating our smooth loss functions is computationally challenging: a naïve algorithm would require $\mathcal{O}(\binom{n}{k})$ operations, where n is the number of classes. Thanks to a connection to polynomial algebra and a divide-and-conquer approach, we provide an algorithm with a time complexity of $\mathcal{O}(kn)$. Furthermore, we present a novel approximation to obtain fast and stable algorithms on GPUs with single floating point precision. We compare the performance of the cross-entropy loss and our margin-based losses in various regimes of noise and data size, for the predominant use case of $k = 5$. Our investigation reveals that our loss is more robust to noise and overfitting than cross-entropy.

2.1 Introduction

In machine learning many classification tasks present inherent label confusion. The confusion can originate from a variety of factors, such as incorrect labeling, incomplete annotation, or some fundamental ambiguities that obfuscate the ground truth label even to a human expert. For example, consider the images from the ImageNet data set (Russakovsky et al., 2015) in Figure 2.1, which illustrate the aforementioned factors. To mitigate these issues, one may require the model to predict the k most likely labels, where k is typically very small compared to the total

number of labels. Then the prediction is considered incorrect if all of its k labels differ from the ground truth, and correct otherwise. This is commonly referred to as the top-k error. Learning such models is a longstanding task in machine learning, and many loss functions for top-k error have been suggested in the literature.

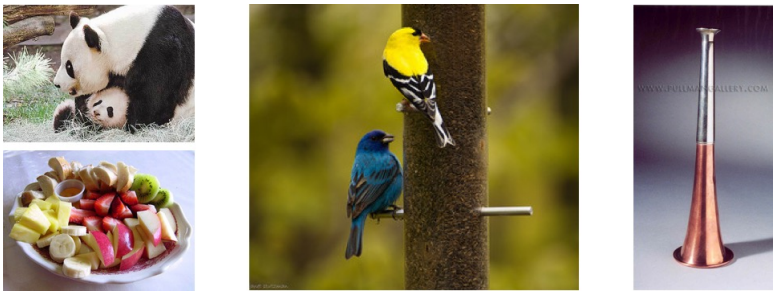


Figure 2.1: *Examples of images with label confusion, from the validation set of ImageNet. The top-left image is incorrectly labeled as "red panda", instead of "giant panda". The bottom-left image is labeled as "strawberry", although the categories "apple", "banana" and "pineapple" would be other valid labels. The center image is labeled as "indigo bunting", which is only valid for the lower bird of the image. The right-most image is labeled as a cocktail shaker, yet could arguably be a part of a music instrument (for example with label "cornet, horn, trumpet, trump"). Such examples motivate the need to predict more than a single label per image.*

In the context of correctly labeled large data, deep neural networks trained with cross-entropy have shown exemplary capacity to accurately approximate the data distribution. An illustration of this phenomenon is the performance attained by deep convolutional neural networks on the ImageNet challenge. Specifically, state-of-the-art models trained with cross-entropy yield remarkable success on the top-5 error, although cross-entropy is not tailored for top-5 error minimization. This phenomenon can be explained by the fact that cross-entropy is top-k calibrated for any k (Lapin et al., 2016), an asymptotic property which is verified in practice in the large data setting. However, in cases where only a limited amount of data is available, learning large models with cross-entropy can be prone to over-fitting on incomplete or noisy labels.

To alleviate the deficiency of cross-entropy, we present a new family of top-k classification loss functions for deep neural networks. Taking inspiration from multi-class SVMs, our loss creates a margin between the correct top-k predictions and the incorrect ones. Our empirical results show that traditional top-k loss functions

do not perform well in combination with deep neural networks. We believe that the reason for this is the lack of smoothness and the sparsity of the derivatives that are used in backpropagation. In order to overcome this difficulty, we smooth the loss with a temperature parameter. The evaluation of the smooth function and its gradient is challenging, as smoothing increases the naïve time complexity from $\mathcal{O}(n)$ to $\mathcal{O}(\binom{n}{k})$. With a connection to polynomial algebra and a divide-and-conquer method, we present an algorithm with $\mathcal{O}(kn)$ time complexity and training time comparable to cross-entropy in practice. We provide insights for numerical stability of the forward pass. To deal with instabilities of the backward pass, we derive a novel approximation. Our investigation reveals that our top-k loss outperforms cross-entropy in the presence of noisy labels or in the absence of large amounts of data. We further confirm that the difference of performance reduces with large correctly labeled data, which is consistent with known theoretical results.

2.2 Related Work

Top-k Loss Functions. The majority of the work on top-k loss functions has been applied to shallow models: Lapin et al. (2016) suggest a convex surrogate on the top-k loss; Fan et al. (2017) select the k largest individual losses in order to be robust to data outliers; Chang et al. (2017) formulate a truncated re-weighted top-k loss as a difference-of-convex objective and optimize it with the Concave-Convex Procedure (Yuille and Rangarajan, 2002); and Yan et al. (2017) propose to use a combination of top-k classifiers and to fuse their outputs.

Closest to our work is the extensive review of top-k loss functions for computer vision by Lapin et al. (2017). The authors conduct a study of a number of top-k loss functions derived from cross-entropy and hinge losses. Interestingly, they prove that for any k, cross-entropy is top-k calibrated, which is a necessary condition for the classifier to be consistent with regard to the theoretically optimal top-k risk. In other words, cross-entropy satisfies an essential property to perform the optimal top-k classification decision for any k in the limit of infinite data. This may explain why cross-entropy performs well on top-5 error on large scale data sets. While thorough,

the experiments are conducted on linear models, or pre-trained deep networks that are fine-tuned. For a more complete analysis, we wish to design loss functions that allow for the training of deep neural networks from a random initialization.

Smoothing. Smoothing is a helpful technique in optimization (Beck and Teboulle, 2012). In work closely related to ours, Lee and Mangasarian (2001) show that smoothing a binary SVM with a temperature parameter improves the theoretical convergence speed of their algorithm. Schwing et al. (2012) use a temperature parameter to smooth latent variables for structured prediction. Lapin et al. (2017) apply Moreau-Yosida regularization to smooth their top-k surrogate losses.

Smoothing has also been applied in the context of deep neural networks. In particular, Zheng et al. (2015) and Clevert et al. (2016) both suggest modifying the non-smooth ReLU activation to improve the training. Gulcehre et al. (2017) suggest to introduce "mollifyers" to smooth the objective function by gradually increasing the difficulty of the optimization problem. Chaudhari et al. (2017) add a local entropy term to the loss to promote solutions with high local entropy. These smoothing techniques are used to speed up the optimization or improve generalization. In this work, we show that smoothing is necessary for the neural network to perform well in combination with our loss function. We hope that this insight can also help the design of losses for tasks other than top-k error minimization.

2.3 Top-k SVM

2.3.1 Background: Multi-Class SVM

In order to build an intuition about top-k losses, we start with the simple case of $k = 1$, namely multi-class classification, where the output space is defined as $\mathcal{Y} = \{1, ..., n\}$. We suppose that a vector of scores per label $\mathbf{s} \in \mathbb{R}^n$, and a ground truth label $y \in \mathcal{Y}$ are both given. The vector $\mathbf{s}$ is the output of the model we wish to learn, for example a linear model or a deep neural network. The notation $\mathbb{1}$ will refer to the indicator function over Boolean statements (1 if true, 0 if false).

Prediction. The prediction is given by any index with maximal score:

$$P(\mathbf{s}) \in \operatorname{argmax} \mathbf{s}. \tag{2.1}$$

Loss. The classification loss incurs a binary penalty by comparing the prediction to the ground truth label. Plugging in equation (2.1), this can also be written in terms of scores $\mathbf{s}$ as follows:

$$\Lambda(\mathbf{s}, y) \triangleq \mathbb{1}(y \neq P(\mathbf{s})) = \mathbb{1}(\max_{j \in \mathcal{Y}} s_j > s_y). \tag{2.2}$$

Surrogate. The loss in equation (2.2) is not amenable to optimization, as it is not even continuous in $\mathbf{s}$. To overcome this difficulty, a typical approach in machine learning is to resort to a surrogate loss that provides a continuous upper bound on Λ. Crammer and Singer (2001) suggest the following upper bound on the loss, known as the multi-class SVM loss:

$$l(\mathbf{s}, y) = \max \left\{ \max_{j \in \mathcal{Y} \setminus \{y\}} \{s_j + 1\} - s_y, 0 \right\}. \tag{2.3}$$

In other words, the surrogate loss is zero if the ground truth score is higher than all other scores by a margin of at least one. Otherwise it incurs a penalty which is linear in the difference between the score of the ground truth and the highest score over all other classes.

Rescaling. Note that the value of 1 as a margin is an arbitrary choice, and can be changed to α for any $\alpha > 0$. This simply entails that we consider the cost Λ of a misclassification to be α instead of 1. Moreover, we show in Proposition 16 of Appendix A.4.2 how the choices of α and of the quadratic regularization hyper-parameter are interchangeable.

2.3.2 Top-k Classification

We now generalize the above framework to top-k classification, where $k \in \{1, ..., n-1\}$. We use the following notation: for $p \in \{1, ..., n\}$, $s_{[p]}$ refers to the p-th largest element of $\mathbf{s}$, and $\mathbf{s}_{\setminus p}$ to the vector $(s_1, ..., s_{p-1}, s_{p+1}, ..., s_n) \in \mathbb{R}^{n-1}$ (that is, the

vector $\mathbf{s}$ with the p-th element omitted). The term $\mathcal{Y}^{(k)}$ denotes the set of k-tuples with k distinct elements of $\mathcal{Y}$. Note that we use a bold font for a tuple $\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}$ in order to distinguish it from a single label $\bar{y} \in \mathcal{Y}$.

Prediction. Given the scores $\mathbf{s} \in \mathbb{R}^n$, the top-$k$ prediction consists of any set of labels corresponding to the k largest scores:

$$P_k(\mathbf{s}) \in \left\{ \bar{\mathbf{y}} \in \mathcal{Y}^{(k)} : \forall\, i \in \{1, .., k\},\ s_{\bar{y}_i} \geq s_{[k]} \right\}. \tag{2.4}$$

Loss. The loss depends on whether y is part of the top-k prediction, which is equivalent to comparing the k-largest score with the ground truth score:

$$\Lambda_k(\mathbf{s}, y) \triangleq \mathbb{1}(y \notin P_k(\mathbf{s})) = \mathbb{1}(s_{[k]} > s_y). \tag{2.5}$$

Again, such a binary loss is not suitable for optimization. Thus we introduce a surrogate loss.

Surrogate. As pointed out in Lapin et al. (2015), there is a natural extension of the previous multi-class case:

$$l_k(\mathbf{s}, y) \triangleq \max \left\{ \left(\mathbf{s}_{\setminus y} + \mathbf{1} \right)_{[k]} - s_y, 0 \right\}. \tag{2.6}$$

This loss creates a margin between the ground truth and the k-th largest score, irrespectively of the values of the $(k-1)$-largest scores. Note that we retrieve the formulation of Crammer and Singer (2001) for $k = 1$.

Difficulty of the Optimization. The surrogate loss l_k of equation (2.6) suffers from two disadvantages that make it difficult to optimize: (i) it is not a smooth function of $\mathbf{s}$ – it is continuous but not differentiable – and (ii) its weak derivatives have at most two non-zero elements. Indeed at most two elements of $\mathbf{s}$ are retained by the $(\cdot)_{[k]}$ and max operators in equation (2.6). All others are discarded and thus get zero derivatives. When l_k is coupled with a deep neural network, the model typically yields poor performance, even on the training set. Similar difficulties to

optimizing a piecewise linear loss have also been reported by Li et al. (2017) in the context of multi-label classification. We illustrate this in the next section.

We postulate that the difficulty of the optimization explains why there has been little work exploring the use of SVM losses in deep learning (even in the case $k = 1$), and that this work may help remedy it. We propose a smoothing that alleviates both issues (i) and (ii), and we present experimental evidence that the smooth surrogate loss offers better performance in practice.

2.3.3 Smooth Surrogate Loss

Reformulation. We introduce the following notation: given a label $\bar{y} \in \mathcal{Y}$, $\mathcal{Y}_{\bar{y}}^{(k)}$ is the subset of tuples from $\mathcal{Y}^{(k)}$ that include $\bar{y}$ as one of their elements. For $\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}$ and $y \in \mathcal{Y}$, we further define $\Delta_k(\bar{\mathbf{y}}, y) \triangleq \mathbb{1}(y \notin \bar{\mathbf{y}})$. Then, by adding and subtracting the $k-1$ largest scores of $\mathbf{s}_{\setminus y}$ as well as s_y, we obtain:

$$\begin{aligned} l_k(\mathbf{s}, y) &= \max\left\{\left(\mathbf{s}_{\setminus y} + \mathbf{1}\right)_{[k]} - s_y, 0\right\}, \\ &= \max_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}} \left\{\Delta_k(\bar{\mathbf{y}}, y) + \sum_{j \in \bar{\mathbf{y}}} s_j\right\} - \max_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \left\{\sum_{j \in \bar{\mathbf{y}}} s_j\right\}. \end{aligned} \tag{2.7}$$

We give a more detailed proof of this in Appendix A.1.1. Since the margin can be rescaled without loss of generality, we rewrite l_k as:

$$l_k(\mathbf{s}, y) = \max_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}} \left\{\Delta_k(\bar{\mathbf{y}}, y) + \frac{1}{k}\sum_{j \in \bar{\mathbf{y}}} s_j\right\} - \max_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \left\{\frac{1}{k}\sum_{j \in \bar{\mathbf{y}}} s_j\right\}. \tag{2.8}$$

Smoothing. In the form of equation (2.8), the loss function can be smoothed with a temperature parameter $\tau > 0$:

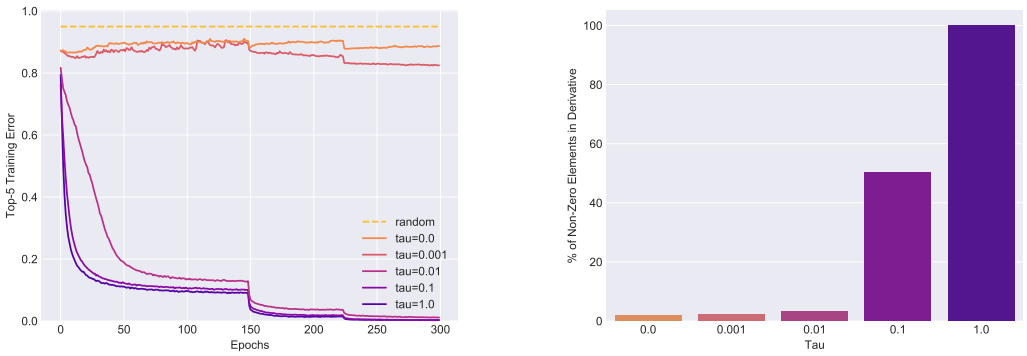
$$\begin{aligned} L_{k,\tau}(\mathbf{s}, y) = \tau \log\left[\sum_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}} \exp\left(\frac{1}{\tau}\left(\Delta_k(\bar{\mathbf{y}}, y) + \frac{1}{k}\sum_{j \in \bar{\mathbf{y}}} s_j\right)\right)\right] \\ - \tau \log\left[\sum_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \exp\left(\frac{1}{k\tau}\sum_{j \in \bar{\mathbf{y}}} s_j\right)\right]. \end{aligned} \tag{2.9}$$

Note that we have changed the notation to use $L_{k,\tau}$ to refer to the smooth loss. In what follows, we first outline the properties of $L_{k,\tau}$ and its relationship with cross-entropy. Then we show the empirical advantage of $L_{k,\tau}$ over its non-smooth counter-part l_k.

Properties of the Smooth Loss. The smooth loss $L_{k,\tau}$ has a few interesting properties. First, for any $\tau > 0$, $L_{k,\tau}$ is infinitely differentiable and has non-sparse gradients. Second, under mild conditions, when $\tau \to 0^+$, the non-maximal terms become negligible, therefore the summations collapse to maximizations and $L_{k,\tau} \to l_k$ in a pointwise sense (Proposition 10 in Appendix A.1.2). Third, $L_{k,\tau}$ is an upper bound on l_k if and only if $k = 1$ (Proposition 11 in Appendix A.1.3), but $L_{k,\tau}$ is, up to a scaling factor, an upper bound on Λ_k (Proposition 12 in Appendix A.1.4). This makes it a valid surrogate loss for the minimization of Λ_k.

Relationship with Cross-Entropy. We have previously seen that the margin can be rescaled by a factor of $\alpha > 0$. In particular, if we scale Δ by $\alpha \to 0^+$ and choose a temperature $\tau = 1$, it can be seen that $L_{1,1}$ becomes exactly the cross-entropy loss for classification. In that sense, $L_{k,\tau}$ is a generalization of the cross-entropy loss to: (i) different values of $k \geq 1$, (ii) different values of temperature and (iii) higher margins with the scaling α of Δ. For simplicity purposes, we will keep $\alpha = 1$ in this work.

Experimental Validation. In order to show how smoothing helps the training, we train a DenseNet 40-12 on CIFAR-100 from Huang et al. (2017) with the same hyper-parameters and learning rate schedule. The only difference with Huang et al. (2017) is that we replace the cross-entropy loss with $L_{5,\tau}$ for different values of τ. We plot the top-5 training error in Figure 2.2a (for each curve, the value of τ is held constant during training):



(a) *Top-5 training error for different values of τ. The dashed line $y = 0.95$ represents the base error for random predictions. The successive drops in the curves correspond to the decreases of the learning rate at epochs 150 and 225.*

(b) *Proportion of non (numerically) zero elements in the loss derivatives for different values of τ. These values are obtained with the initial random weights of the neural network, and are averaged over the training set.*

Figure 2.2: *Influence of the temperature τ on the learning of a DenseNet 40-12 on CIFAR-100. We confirm that smoothing helps the training of a neural network in Figure 2.2a, where a large enough value of τ greatly helps the performance on the training set. In Figure 2.2b, we observe that such high temperatures yield gradients that are not sparse. In other words, with a high temperature, the gradient is informative about a greater number of labels, which helps the training of the model.*

We remark that the network exhibits good accuracy when τ is high enough (0.01 or larger). For τ too small, the model fails to converge to a good critical point. When τ is positive but small, the function is smooth but the gradients are numerically sparse (see Figure 2.2b), which suggests that the smoothness property is not sufficient and that non-sparsity is a key factor here.

2.4 Computational Challenges and Efficient Algorithms

2.4.1 Challenge

Experimental evidence suggests that it is beneficial to use $L_{k,\tau}$ rather than l_k to train a neural network. However, at first glance, $L_{k,\tau}$ may appear prohibitively expensive to compute. Specifically, there are summations over $\mathcal{Y}^{(k)}$ and $\mathcal{Y}_y^{(k)}$, which have a cardinality of $\binom{n}{k}$ and $\binom{n}{k-1}$ respectively. For instance for ImageNet, we have $k = 5$ and $n = 1,000$, which amounts to $\binom{n}{k} \simeq 8.10^{12}$ terms to compute

and sum over for each single sample, thereby making the approach practically infeasible. This is in stark contrast with l_k, for which the most expensive operation is to compute the k-th largest score of an array of size n, which can be done in $\mathcal{O}(n)$. To overcome this computational challenge, we will now reframe the problem and reveal its exploitable structure.

For a vector $\mathbf{e} \in \mathbb{R}^n$ and $i \in \{1, .., n\}$, we define $\sigma_i(\mathbf{e})$ as the sum of all products of i distinct elements of $\mathbf{e}$. Explicitly, $\sigma_i(\mathbf{e})$ can be written as $\sigma_i(\mathbf{e}) = \sum_{1 \leq j_1 < \ldots < j_i \leq n} e_{j_1} ... e_{j_i}$. The terms σ_i are known as the elementary symmetric polynomials. We further define $\sigma_0(\mathbf{e}) = 1$ for convenience.

We now re-write $L_{k,\tau}$ using the elementary symmetric polynomials, which appear naturally when separating the terms that contain the ground truth from the ones that do not:

$$
\begin{aligned}
L_{k,\tau}(\mathbf{s}, y) &= \tau \log \left[\sum_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}} \exp\left(\Delta_k(\bar{\mathbf{y}}, y)/\tau\right) \prod_{j \in \bar{\mathbf{y}}} \exp(s_j/k\tau) \right] \\
&\quad - \tau \log \left[\sum_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \prod_{j \in \bar{\mathbf{y}}} \exp(s_j/k\tau) \right], \\
&= \tau \log \left[\sum_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \prod_{j \in \bar{\mathbf{y}}} \exp(s_j/k\tau) + \exp\left(1/\tau\right) \sum_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)} \setminus \mathcal{Y}_y^{(k)}} \prod_{j \in \bar{\mathbf{y}}} \exp(s_j/k\tau) \right] \\
&\quad - \tau \log \left[\sum_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \prod_{j \in \bar{\mathbf{y}}} \exp(s_j/k\tau) \right], \qquad (2.10) \\
&= \tau \log \left[\exp(s_y/k\tau) \sigma_{k-1}(\exp(\mathbf{s}_{\setminus y}/k\tau)) + \exp\left(1/\tau\right) \sigma_k(\exp(\mathbf{s}_{\setminus y}/k\tau)) \right] \\
&\quad - \tau \log \left[\exp(s_y/k\tau) \sigma_{k-1}(\exp(\mathbf{s}_{\setminus y}/k\tau)) \right].
\end{aligned}
$$

Note that the application of exp to vectors is meant in an element-wise fashion. The last equality of equation (2.10) reveals that the challenge is to efficiently compute σ_{k-1} and σ_k, and their derivatives for the optimization.

While there are existing algorithms to evaluate the elementary symmetric polynomials, they have been designed for computations on CPU with double floating point precision. For the most recent work, see Jiang et al. (2016). To efficiently train deep neural networks with $L_{k,\tau}$, we need algorithms that are numerically

stable with single floating point precision and that exploit GPU parallelization. In the next sections, we design algorithms that meet these requirements. The final performance is compared to the standard alternative algorithm in Appendix A.2.3.

2.4.2 Forward Computation

We consider the general problem of efficiently computing (σ_{k-1}, σ_k). Our goal is to compute $\sigma_k(\mathbf{e})$, where $\mathbf{e} \in \mathbb{R}^n$ and $k \ll n$. Since this algorithm will be applied to $\mathbf{e} = \exp(\mathbf{s}_{\setminus y}/k\tau)$ (see equation (2.10)), we can safely assume $e_i \neq 0$ for all $i \in [\![1, n]\!]$.

The main insight of our approach is the connection of $\sigma_i(\mathbf{e})$ to the polynomial:

$$P \triangleq (X + e_1)(X + e_2)...(X + e_n). \tag{2.11}$$

Indeed, if we expand P to $\alpha_0 + \alpha_1 X + ... + \alpha_n X^n$, Vieta's formula gives the relationship:

$$\forall i \in [\![0, n]\!], \quad \alpha_i = \sigma_{n-i}(\mathbf{e}). \tag{2.12}$$

Therefore, it suffices to compute the coefficients α_{n-k} to obtain the value of $\sigma_k(\mathbf{e})$. To compute the expansion of P, we can use a divide-and-conquer approach with polynomial multiplications when merging two branches of the recursion.

This method computes all $(\sigma_i)_{1 \leq i \leq n}$ instead of the only $(\sigma_i)_{k-1 \leq i \leq k}$ that we require. Since we do not need $\sigma_i(\mathbf{e})$ for $i > k$, we can avoid computations of all coefficients for a degree higher than $n - k$. However, typically $k \ll n$. For example, in ImageNet, we have $k = 5$ and $n = 1,000$, therefore we have to compute coefficients up to a degree 995 instead of 1,000, which is a negligible improvement. To turn $k \ll n$ to our advantage, we notice that $\sigma_i(\mathbf{e}) = \sigma_n(\mathbf{e})\sigma_{n-i}(1/\mathbf{e})$. Moreover, $\sigma_n(\mathbf{e}) = \prod_{i=1}^{n} e_i$ can be computed in $\mathcal{O}(n)$. Therefore we introduce the polynomial:

$$Q \triangleq \sigma_n(\mathbf{e})(X + \frac{1}{e_1})(X + \frac{1}{e_2})...(X + \frac{1}{e_n}). \tag{2.13}$$

Then if we expand Q to $\beta_0 + \beta_1 X + ... + \beta_n X^n$, we obtain with Vieta's formula again:

$$\forall i \in [\![0, n]\!], \quad \beta_i = \sigma_n(\mathbf{e})\sigma_{n-i}(1/\mathbf{e}) = \sigma_i(\mathbf{e}). \tag{2.14}$$

Subsequently, in order to compute $\sigma_k(\mathbf{e})$, we only require the k first coefficients of Q, which is very efficient when k is small in comparison with n. This results in a time complexity of $\mathcal{O}(kn)$ (Proposition 13 in Appendix A.2.1). Moreover, there are only $\mathcal{O}(\log(n))$ levels of recursion, and since every level can have its operations parallelized, the resulting algorithm scales very well with n when implemented on a GPU (see Appendix A.2.3 for practical runtimes).

The algorithm is described in Algorithm 1: step 2 initializes the polynomials for the divide and conquer method. While the polynomial has not been fully expanded, steps 5-6 merge branches by performing the polynomial multiplications (which can be done in parallel). Step 10 adjusts the coefficients using equation (2.14). We point out that we could obtain an algorithm with a time complexity of $\mathcal{O}(n\log(k)^2)$ if we were using Fast Fourier Transform for polynomial multiplications in steps 5-6. Since we are interested in the case where k is small (typically 5), such an improvement is negligible.

Algorithm 1 *Forward Pass*

Require: $\mathbf{e} \in (\mathbb{R}_+^*)^n$, $k \in \mathbb{N}^*$

1: $t \leftarrow 0$
2: $P_i^{(t)} \leftarrow (1, 1/e_i)$ for $i \in [\![1, n]\!]$ ▷ Initialize n polynomials to $X + \frac{1}{e_i}$ (encoded by coefficients)
3: $p \leftarrow n$ ▷ Number of polynomials
4: **while** $p > 1$ **do** ▷ Merge branches with polynomial multiplications
5: $\quad P_1^{(t+1)} \leftarrow P_1^{(t)} * P_2^{(t)}$ ▷ Polynomial multiplication up to degree k
$\quad \ldots$
6: $\quad P_{(p-1)//2}^{(t+1)} \leftarrow P_{p-1}^{(t)} * P_p^{(t)}$ ▷ Polynomial multiplication up to degree k
7: $\quad t \leftarrow t + 1$
8: $\quad p \leftarrow (p-1)//2$ ▷ Update number of polynomials
9: **end while**
10: $P^{(t+1)} \leftarrow P^{(t)} \times \prod_{i=1}^{n} e_i$ ▷ Recover $\sigma_i(\mathbf{e}) = \sigma_{n-i}(1/\mathbf{e})\sigma_n(\mathbf{e})$
11: **return** $P^{(t+1)}$

Obtaining numerical stability in single floating point precision requires special attention: the use of exponentials with an arbitrarily small temperature parameter is fundamentally unstable. In Appendix A.2.2, we describe how operating in the log-space and using the log-sum-exp trick alleviates this issue. The stability of the resulting algorithm is empirically verified in Appendix A.2.3.

2.4.3 Backward Computation

A side effect of using Algorithm 1 is that a large number of buffers are allocated for automatic differentiation: for each addition in log-space, we apply log and exp operations, each of which needs to store values for the backward pass. This results in a significant amount of time spent on memory allocations, which become the time bottleneck. To avoid this, we exploit the structure of the problem and design a backward algorithm that relies on the results of the forward pass. By avoiding the memory allocations and considerably reducing the number of operations, the backward pass is then sped up by one to two orders of magnitude and becomes negligible in comparison to the forward pass. We describe our efficient backward pass in more details below.

First, we introduce the notation for derivatives:

$$\text{For } i \in [\![1, n]\!], 1 \leq j \leq k, \quad \delta_{j,i} \triangleq \frac{\partial \sigma_j(\mathbf{e})}{\partial e_i}. \tag{2.15}$$

We now observe that:

$$\delta_{j,i} = \sigma_{j-1}(\mathbf{e}_{\setminus i}). \tag{2.16}$$

In other words, equation (2.16) states that $\delta_{j,i}$, the derivative of $\sigma_j(\mathbf{e})$ with respect to e_i, is the sum of product of all $(j-1)$-tuples that do not include e_i. One way of obtaining $\sigma_{j-1}(\mathbf{e}_{\setminus i})$ is to compute a forward pass for $\mathbf{e}_{\setminus i}$, which we would need to do for every $i \in [\![1, n]\!]$. To avoid such expensive computations, we remark that $\sigma_j(\mathbf{e})$ can be split into two terms: the ones that contain e_i (which can expressed as $e_i \sigma_{j-1}(\mathbf{e}_{\setminus i})$) and the ones that do not (which are equal to $\sigma_j(\mathbf{e}_{\setminus i})$ by definition). This gives the following relationship:

$$\sigma_j(\mathbf{e}_{\setminus i}) = \sigma_j(\mathbf{e}) - e_i \sigma_{j-1}(\mathbf{e}_{\setminus i}). \tag{2.17}$$

Simplifying equation (2.17) using equation (2.16), we obtain the following recursive relationship:

$$\delta_{j,i} = \sigma_{j-1}(\mathbf{e}) - e_i \delta_{j-1,i}. \tag{2.18}$$

Since the $(\sigma_j(\mathbf{e}))_{1\leq i\leq k}$ have been computed during the forward pass, we can initialize the induction with $\delta_{1,i} = 1$ and iteratively compute the derivatives $\delta_{j,i}$ for $j \geq 2$ with equation (2.18). This is summarized in Algorithm 2.

Algorithm 2 *Backward Pass*

Require: $\mathbf{e}$, $(\sigma_j(\mathbf{e}))_{1\leq j\leq k}$, $k \in \mathbb{N}^*$ ▷ $(\sigma_j(\mathbf{e}))_{1\leq j\leq k}$ have been computed in the forward pass
$\delta_{1,i} = 1$ for $i \in [\![1, n]\!]$
for $j \in [\![1, k]\!]$ **do**
    $\delta_{j,i} = \sigma_{j-1}(\mathbf{e}) - e_i\delta_{j-1,i}$ for $i \in [\![1, n]\!]$
end for

Algorithm 2 is subject to numerical instabilities (Observation 2 in Appendix A.2.2). In order to avoid these, one solution is to use equation (2.16) for each unstable element, which requires numerous forward passes. To avoid this inefficiency, we provide a novel approximation in Appendix A.2.2: the computation can be stabilized by an approximation with significantly smaller overhead.

2.5 Experiments

Theoretical results suggest that Cross-Entropy (CE) is an optimal classifier in the limit of infinite data, by accurately approximating the data distribution. In practice, the presence of label noise makes the data distribution more complex to estimate when only a finite number of samples is available. For these reasons, we explore the behavior of CE and $L_{k,\tau}$ when varying the amount of label noise and the training data size. For the former, we introduce label noise in the CIFAR-100 data set (Krizhevsky, 2009) in a manner that would not perturb the top-5 error of a perfect classifier. For the latter, we vary the training data size on subsets of the ImageNet data set (Russakovsky et al., 2015).

In all the following experiments, the temperature parameter is fixed throughout training. This choice is discussed in Appendix A.4.1. The algorithms are implemented in PyTorch (Paszke et al., 2017) and are publicly available at `https://github.com/oval-group/smooth-topk`. Experiments on CIFAR-100 and ImageNet are performed on respectively one and two Nvidia Titan Xp cards.

2.5.1 CIFAR-100 with noise

Data set. In this experiment, we investigate the impact of label noise on CE and $L_{5,1}$. The CIFAR-100 data set contains 60,000 RGB images, with 50,000 samples for training-validation and 10,000 for testing. There are 20 "coarse" classes, each consisting of 5 "fine" labels. For example, the coarse class "people" is made up of the five fine labels "baby", "boy", "girl", "man" and "woman". In this set of experiments, the images are centered and normalized channel-wise before they are fed to the network. We use the standard data augmentation technique with random horizontal flips and random crops of size 32×32 on the images padded with 4 pixels on each side.

We introduce noise in the labels as follows: with probability p, each fine label is replaced by a fine label from the same coarse class. This new label is chosen at random and may be identical to the original label. Note that all instances generated by data augmentation from a single image are assigned the same label. The case $p = 0$ corresponds to the original data set without noise, and $p = 1$ to the case where the label is completely random (within the fine labels of the coarse class). With this method, a perfect top-5 classifier would still be able to achieve 100 % accuracy by systematically predicting the five fine labels of the unperturbed coarse label.

Methods. To evaluate our loss functions, we use the architecture DenseNet 40-40 from Huang et al. (2017), and we use the same hyper-parameters and learning rate schedule as in Huang et al. (2017). The temperature parameter is fixed to one. When the level of noise becomes non-negligible, we empirically find that CE suffers from over-fitting and significantly benefits from early stopping – which our loss does not need. Therefore we help the baseline and hold out a validation set of 5,000 images, on which we monitor the accuracy across epochs. Then we use the model with the best top-5 validation accuracy and report its performance on the test set. Results are averaged over three runs with different random seeds.

Table 2.1: *Testing performance on CIFAR-100 with different levels of label noise. With noisy labels, $L_{5,1}$ consistently outperforms CE on both top-5 and top-1 accuracies, with improvements increasingly significant with the level of noise. For reference, a model making random predictions would obtain 1% top-1 accuracy and 5% top-5 accuracy.*

Noise Level	Top-1 Accuracy (%)		Top-5 Accuracy (%)	
	CE	$L_{5,1}$	CE	$L_{5,1}$
0.0	**76.68**	69.33	**94.34**	94.29
0.2	68.20	**71.30**	87.89	**90.59**
0.4	61.18	**70.02**	83.04	**87.39**
0.6	52.50	**67.97**	79.59	**83.86**
0.8	35.53	**55.85**	74.80	**79.32**
1.0	14.06	**15.28**	67.70	**72.93**

Results. As seen in Table 2.1, $L_{5,1}$ outperforms CE on the top-5 testing accuracy when the labels are noisy, with an improvement of over 5% in the case $p = 1$. When there is no noise in the labels, CE provides better top-1 performance, as expected. It also obtains a better top-5 accuracy, although by a very small margin. Interestingly, $L_{5,1}$ outperforms CE on the top-1 error when there is noise, although $L_{5,1}$ is not a surrogate for the top-1 error. For example when $p = 0.8$, $L_{5,1}$ still yields an accuracy of 55.85%, as compared to 35.53% for CE. This suggests that when the provided label is only informative about top-5 predictions (because of noise or ambiguity), it is preferable to use $L_{5,1}$.

2.5.2 ImageNet

Data set. As shown in Figure 2.1, the ImageNet data set presents different forms of ambiguity and noise in the labels. It also has a large number of training samples, which allows us to explore different regimes up to the large-scale setting. Out of the 1.28 million training samples, we use subsets of various sizes and always hold out a balanced validation set of 50,000 images. We then report results on the 50,000 images of the official validation set, which we use as our test set. Images are resized so that their smaller dimension is 256, and they are centered and normalized channel-wise. At training time, we take random crops of 224×224 and randomly flip the images horizontally. At test time, we use the standard ten-crop procedure (Krizhevsky et al., 2012).

We report results for the following subset sizes of the data: 64k images (5%), 128k images (10%), 320k images (25%), 640k images (50%) and finally the whole data set (1.28M − 50k = 1.23M images for training). Each strict subset has all 1,000 classes and a balanced number of images per class. The largest subset has the same slight unbalance as the full ImageNet data set.

Methods. In all the following experiments, we train a ResNet-18 (He et al., 2016), adapting the protocol of the ImageNet experiment in Huang et al. (2017). In more details, we optimize the model with Stochastic Gradient Descent with a batch-size of 256, for a total of 120 epochs. We use a Nesterov momentum of 0.9. The temperature is set to 0.1 for the SVM loss (we discuss the choice of the temperature parameter in Appendix A.4.1). The learning rate is divided by ten at epochs 30, 60 and 90, and is set to an initial value of 0.1 for CE and 1 for $L_{5,0.1}$. The quadratic regularization hyper-parameter is set to 0.0001 for CE. For $L_{5,0.1}$, it is set to 0.000025 to preserve a similar relative weighting of the loss and the regularizer. For both methods, training on the whole data set takes about a day and a half (it is only 10% longer with $L_{5,0.1}$ than with CE). As in the previous experiments, the validation top-5 accuracy is monitored at every epoch, and we use the model with best top-5 validation accuracy to report its test error.

Probabilities for Multiple Crops. Using multiple crops requires a probability distribution over labels for each crop. Then this probability is averaged over the crops to compute the final prediction. The standard method is to use a softmax activation over the scores. We believe that such an approach is only grounded to make top-1 predictions. The probability of a label $\bar{y}$ being part of the top-5 prediction should be marginalized over all combinations of 5 labels that include $\bar{y}$ as one of their elements. This can be directly computed with our algorithms to evaluate σ_k and its derivative. We refer the reader to Appendix A.3 for details. All the reported results of top-5 error with multiple crops are computed with this method. This provides a systematic boost of at least 0.2% for all loss functions. In fact, it is more beneficial to the CE baseline, by up to 1% in the small data setting.

Table 2.2: *Top-5 accuracy (%) on ImageNet using training sets of various sizes. Results are reported on the official validation set, which we use as our test set.*

% Data Set	Number of Images	CE	$L_{5,0.1}$
100%	1.23M	**90.67**	90.61
50%	640k	87.57	**87.87**
25%	320k	82.62	**83.38**
10%	128k	71.06	**73.10**
5%	64k	58.31	**60.44**

Results. The results of Table 2.2 confirm that $L_{5,0.1}$ offers better top-5 error than CE when the amount of training data is restricted. As the data set size increases, the difference of performance becomes very small, and CE outperforms $L_{5,0.1}$ by an insignificant amount in the full data setting.

2.6 Conclusion

This work has introduced a new family of loss functions for the direct minimization of the top-k error (that is, without the need for fine-tuning). We have empirically shown that non-sparsity is essential for loss functions to work well with deep neural networks Thanks to a connection to polynomial algebra and a novel approximation, we have presented efficient algorithms to compute the smooth loss and its gradient. The experimental results have demonstrated that our smooth top-5 loss function is more robust to noise and overfitting than cross-entropy when the amount of training data is limited.

We have argued that smoothing the surrogate loss function helps the training of deep neural networks. This insight is not specific to top-k classification, and we hope that it will help the design of other surrogate loss functions. In particular, structured prediction problems could benefit from smoothed SVM losses. How to efficiently compute such smooth functions could open interesting research problems.

Acknowledgments

This work was supported by the EPSRC grants AIMS CDT EP/L015987/1, Seebibyte EP/M013774/1, EP/P020658/1 and TU/B/000048, and by YouGov. Many thanks to A. Desmaison and R. Bunel for the helpful discussions.

3

Trusting SVMs for Piecewise Linear CNNs

Contents

Abstract

We present a novel layerwise optimization algorithm for the learning objective of Piecewise-Linear Convolutional Neural Networks (PL-CNNs), a large class of convolutional neural networks. Specifically, PL-CNNs employ piecewise linear non-linearities such as the commonly used ReLU and max-pool, and an SVM classifier as the final layer. The key observation of our approach is that the problem corresponding to the parameter estimation of a layer can be formulated as a difference-of-convex (DC) program, which happens to be a latent structured SVM. We optimize the DC program using the concave-convex procedure, which requires us to iteratively solve a structured SVM problem. This allows to design an optimization algorithm with an optimal learning rate that does not require any tuning. Using the MNIST, CIFAR and ImageNet data sets, we show that our approach always improves over the state of the art variants of backpropagation and scales to large data and large network settings.

3.1 Introduction

The backpropagation algorithm is commonly employed to estimate the parameters of a convolutional neural network (CNN) using a supervised training data set (Rumelhart et al., 1986). Part of the appeal of backpropagation comes from the fact that it is applicable to a wide variety of networks, namely those that have (sub-)differentiable non-linearities and employ a (sub-)differentiable learning objective. However, the generality of backpropagation comes at the cost of a high sensitivity to its hyperparameters such as the learning rate and momentum. Standard line-search algorithms cannot be used on the primal objective function in this setting, as (i) there may not exist a step-size guaranteeing a monotonic decrease because of the use of sub-gradients, and (ii) even in the smooth case, each function evaluation requires a forward pass over the entire data set without any update, making the approach computationally unfeasible. Choosing the learning rate thus remains an open issue, with the state-of-the-art algorithms suggesting

adaptive learning rates (Duchi et al., 2011; Zeiler, 2012; Kingma and Ba, 2015). In addition, techniques such as batch normalization (Ioffe and Szegedy, 2015) and dropout (Srivastava et al., 2014) have been introduced to respectively reduce the sensitivity to the learning rate and to prevent from overfitting.

With this work, we open a different line of inquiry, namely, is it possible to design more robust optimization algorithms for special but useful classes of CNNs? To this end, we focus on the networks that are commonly used in computer vision. Specifically, we consider CNNs with convolutional and dense layers that apply a set of piecewise linear (PL) non-linear operations to obtain a discriminative representation of an input image. While this assumption may sound restrictive at first, we show that commonly used non-linear operations such as ReLU and max-pool fall under the category of PL functions. The representation obtained in this way is used to classify the image via a multi-class SVM, which forms the final layer of the network. We refer to this class of networks as PL-CNN.

We design a novel, principled algorithm to optimize the learning objective of a PL-CNN. Our algorithm is a layerwise method, that is, it iteratively updates the parameters of one layer while keeping the other layers fixed. For this work, we use a simple schedule over the layers, namely, repeated passes from the output layer to the input one. However, it may be possible to further improve the accuracy and efficiency of our algorithm by designing more sophisticated scheduling strategies. The key observation of our approach is that the parameter estimation of one layer of PL-CNN can be formulated as a difference-of-convex (DC) program that can be viewed as a latent structured SVM problem (Yu and Joachims, 2009). This allows us to solve the DC program using the concave-convex procedure (CCCP) (Yuille and Rangarajan, 2002). Each iteration of CCCP requires us to solve a convex structured SVM problem. To this end, we use the powerful block-coordinate Frank-Wolfe (BCFW) algorithm (Lacoste-Julien and Jaggi, 2013), which solves the dual of the convex program iteratively by computing the conditional gradients corresponding to a subset of training samples. In order to further improve BCFW for PL-CNNs, we extend it in three important ways. First, we introduce a trust-region term that

allows us to initialize the BCFW algorithm using the current estimate of the layer parameters. Second, we reduce the memory requirement of BCFW by an order of magnitude, via an efficient representation of the feature vectors corresponding to the dense layers. Third, we show that, empirically, the number of constraints of the structural SVM problem can be reduced substantially without any loss in accuracy, which allows us to significantly reduce its time complexity.

Compared to backpropagation (Rumelhart et al., 1986) or its variants (Duchi et al., 2011; Zeiler, 2012; Kingma and Ba, 2015), our algorithm offers three advantages. First, the CCCP algorithm provides a monotonic decrease in the learning objective at each layer. Since layerwise optimization itself can be viewed as a block-coordinate method, our algorithm guarantees a monotonic decrease of the overall objective function after each layer's parameters have been updated. Second, since the dual of the SVM problem is a smooth convex quadratic program, each step of the BCFW algorithm (in the inner iteration of the CCCP) provides a monotonic increase in its dual objective. Third, since the only step-size required in our approach comes while solving the SVM dual, we can use the optimal step-size that is computed analytically during each iteration of BCFW (Lacoste-Julien and Jaggi, 2013). In other words, our algorithm has no learning rate, initial or not, that requires tuning.

Using standard network architectures and publicly available data sets, we show that our algorithm provides a boost over the state of the art variants of backpropagation for learning PL-CNNs and we demonstrate scalability of the method.

3.2 Related Work

While some of the early successful approaches for the optimization of deep neural networks relied on greedy layer-wise training (Hinton et al., 2006; Bengio et al., 2007), most currently used methods are variants of backpropagation (Rumelhart et al., 1986) with adaptive learning rates, as discussed in the introduction.

At every iteration, backpropagation performs a forward pass and a backward pass on the network, and updates the parameters of each layer by stochastic or mini-batch gradient descent. This makes the choice of the learning rate critical

for efficient optimization. (Duchi et al., 2011) have proposed the Adagrad convex solver, which adapts the learning rate for every direction and takes into account past updates. Adagrad changes the learning rate to favor steps in gradient directions that have not been observed frequently in past updates. When applied to the non-convex CNN optimization problem, Adagrad may converge prematurely due to a rapid decrease in the learning rate (Goodfellow et al., 2016). In order to prevent this behavior, the Adadelta algorithm (Zeiler, 2012) makes the decay of the learning rate slower. It is worth noting that this fix is empirical, and to the best of our knowledge, provides no theoretical guarantees. (Kingma and Ba, 2015) propose a different scheme for the learning rate, called Adam, which uses an online estimation of the first and second moments of the gradients to provide centered and normalized updates. However all these methods still require the tuning of the initial learning rate to perform well.

Second-order and natural gradient optimization methods have also been a subject of attention. The focus in this line of work has been to come up with appropriate approximations to make the updates cheaper. (Martens and Sutskever, 2012) suggested a Hessian-free second order optimization using finite differences to approximate the Hessian and conjugate gradient to compute the update. (Martens and Grosse, 2015) derive an approximation of the Fisher matrix inverse, which provides a more efficient method for natural gradient descent. (Ollivier, 2013) explore a set of Riemannian methods based on natural gradient descent and quasi-Newton methods to guarantee reparameterization invariance of the problem. (Desjardins et al., 2015) demonstrate a scaled up natural gradient descent method by training on the ImageNet data set (Russakovsky et al., 2015). Though providing more informative updates and solid theoretical support than SGD-based approaches, these methods do not take into account the structure of the problem offered by the commonly used non-linear operations.

Our work is also related to some of the recent developments in optimization for deep learning. For example, (Taylor et al., 2016) use ADMM for massive distribution of computation in a layer-wise fashion, and in particular their method

will yield closed-form updates for any PL-CNN. (Lee et al., 2015) propose to use targets instead of gradients to propagate information through the network, which could help to extend our algorithm. (Zhang et al., 2017b) derive a convex relaxation for the learning objective for a restricted class of CNNs, which also relies on solving an approximate convex problem. In (Amos et al., 2017), the authors identify convex problems for the inference task, when the neural network is a convex function of some of its inputs.

With a more theoretical approach, (Goel et al., 2017) propose an algorithm to learn shallow ReLU nets with guarantees of time convergence and generalization error. (Heinemann et al., 2016) show that a subclass of neural networks can be modeled as an improper kernel, which then reduces the learning problem to a simple SVM with the constructed kernel.

More generally, we believe that our hitherto unknown observation regarding the relationship between PL-CNNs and latent SVMs can (i) allow the progress made in one field to be transferred to the other and (ii) help design a new generation of principled algorithms for deep learning optimization.

3.3 Piecewise Linear Convolutional Neural Networks

A piecewise linear convolutional neural network (PL-CNN) consists of a series of convolutional layers, followed by a series of dense layers, which provides a concise representation of an input image. Each layer of the network performs two operations: a linear transformation (that is, a convolution or a matrix multiplication), followed by a piecewise linear non-linear operation such as ReLU or max-pool. The resulting representation of the image is used for classification via an SVM. In the remainder of this section, we provide a formal description of PL-CNN.

Piecewise Linear Functions. A (continuous) piecewise linear (PL) function $f(\mathbf{u})$ is a function of the following form (Melzer, 1986):

$$f(\mathbf{u}) = \max_{i \in [m]} \{\mathbf{a}_i^\top \mathbf{u}\} - \max_{j \in [n]} \{\mathbf{b}_j^\top \mathbf{u}\}, \tag{3.1}$$

where $[m] = \{1, \cdots, m\}$, and $[n] = \{1, \cdots, n\}$. Each of the two maxima above is a convex function, therefore such a function f is not generally convex, but it is rather a difference of two convex functions. Importantly, many commonly used non-linear operations such as ReLU or max-pool are PL functions of their input. For example, ReLU corresponds to the function $R(v) = \max\{v, 0\}$ where v is a scalar. Similarly, max-pool for a D-dimensional vector $\mathbf{u}$ corresponds to $M(\mathbf{u}) = \max_{i \in [D]}\{\mathbf{e}_i^\top \mathbf{u}\}$, where $\mathbf{e}_i$ is a vector whose i-th element is 1 and all other elements are 0. Given a value of $\mathbf{u}$, we say that (i^*, j^*) is the activation of the PL function at $\mathbf{u}$ if $i^* = \text{argmax}_{i \in [m]}\{\mathbf{a}_i^\top \mathbf{u}\}$ and $j^* = \text{argmax}_{j \in [n]}\{\mathbf{b}_j^\top \mathbf{u}\}$.

PL-CNN Parameters. We denote the parameters of an L layer PL-CNN by $\mathcal{W} = \{W^l; l \in [L]\}$. In other words, the parameters of the l-th layer is defined as W^l. The CNN defines a composite function, that is, the output $\mathbf{z}^{l-1}$ of layer $l-1$ is the input to the layer l. Given the input $\mathbf{z}^{l-1}$ to layer l, the output is computed as $\mathbf{z}^l = \sigma^l(W^l \cdot \mathbf{z}^{l-1})$, where "$\cdot$" is either a convolution or a matrix multiplication, and σ^l is a PL non-linear function, such as ReLU or max-pool. The input to the first layer is an image $\mathbf{x}$, that is, $\mathbf{z}^0 = \mathbf{x}$. We denote the input to the final layer by $\mathbf{z}^L = \Phi(\mathbf{x}; \mathcal{W}) \in \mathbb{R}^D$. In other words, given an image $\mathbf{x}$, the convolutional and dense layers of a PL-CNN provide a D-dimensional representation of $\mathbf{x}$ to the final classification layer. The final layer of a PL-CNN is a C class SVM W^{svm}, which specifies one parameter $W_y^{\text{svm}} \in \mathbb{R}^D$ for each class $y \in \mathcal{Y}$.

Prediction. Given an image $\mathbf{x}$, a PL-CNN predicts its class using the following rule:

$$y^* = \underset{y \in \mathcal{Y}}{\text{argmax}}\, W_y^{\text{svm}} \Phi(\mathbf{x}; \mathcal{W}). \tag{3.2}$$

In other words, the dot product of the D-dimensional representation of $\mathbf{x}$ with the SVM parameter for a class y provides the score for the class. The desired prediction is obtained by maximizing the score over all possible classes.

Learning Objective. Given a training data set $\mathcal{D} = \{(\mathbf{x}_i, y_i), i \in [N]\}$, where $\mathbf{x}_i$ is the input image and y_i is its ground-truth class, we wish to estimate the parameters $\mathcal{W} \cup W^{\text{svm}}$ of the PL-CNN. To this end, we minimize a regularized upper bound on the empirical risk. The risk of a prediction y_i^* given the ground-truth y_i is measured with a user-specified loss function $\Delta(y_i^*, y_i)$. For example, the standard $0-1$ loss has a value of 0 for a correct prediction and 1 for an incorrect prediction. Formally, the parameters of a PL-CNN are estimated using the following learning objective:

$$\min_{\mathcal{W}, W^{\text{svm}}} \frac{\lambda}{2} \sum_{l \in [L] \cup \{\text{svm}\}} \|W^l\|_F^2 + \frac{1}{N} \sum_{i=1}^{N} \max_{\bar{y}_i \in \mathcal{Y}} \left(\Delta(\bar{y}_i, y_i) + \left(W_{\bar{y}_i}^{\text{svm}} - W_{y_i}^{\text{svm}} \right)^T \Phi(\mathbf{x}_i; \mathcal{W}) \right). \tag{3.3}$$

The hyperparameter λ denotes the relative weight of the regularization compared to the upper bound of the empirical risk. Note that, due to the presence of piecewise linear non-linearities, the representation $\Phi(\cdot; \mathcal{W})$ (and hence, the above objective) is highly non-convex in the PL-CNN parameters.

3.4 Parameter Estimation for PL-CNN

In order to enable layerwise optimization of PL-CNNs, we show that parameter estimation of a layer can be formulated as a difference-of-convex (DC) program (subsection 3.4.1). This allows us to use the concave-convex procedure, which solves a series of convex optimization problems (subsection 3.4.2). We show that each convex problem closely resembles a structured SVM objective, which can be addressed by the powerful block-coordinate Frank-Wolfe (BCFW) algorithm. We extend BCFW to improve its initialization, time complexity and memory requirements, thereby enabling its use in learning PL-CNNs (subsection 3.4.3). For the sake of clarity, we only provide sketches of the proofs for those propositions that are necessary for understanding the paper. The detailed proofs of the remaining propositions are provided in the Appendix.

3.4.1 Layerwise Optimization as a DC Program

Given the values of the parameters for the convolutional and the dense layers (that is, $\mathcal{W}$), the learning objective (3.3) is the standard SVM problem in parameters W^{svm}. In other words, it is a convex optimization problem with several efficient solvers (Tsochantaridis et al., 2004; Joachims et al., 2009; Shalev-Shwartz et al., 2009), including the BCFW algorithm (Lacoste-Julien and Jaggi, 2013). Hence, the optimization of the final layer is a computationally easy problem. In contrast, the optimization of the parameters of a convolutional or a dense layer l does not result in a convex program. In general, this problem can be arbitrarily hard to solve. However, in the case of PL-CNN, we show that the problem can be formulated as a specific type of DC program, which enables efficient optimization via the iterative use of BCFW. The key property that enables our approach is the following proposition that shows that the composition of PL functions is also a PL function.

Proposition 1. *Consider PL functions $g : \mathbb{R}^m \to \mathbb{R}$ and $g_i : \mathbb{R}^n \to \mathbb{R}$, for all $i \in [m]$. Define a function $f : \mathbb{R}^n \to \mathbb{R}$ as $f(\mathbf{u}) = g([g_1(\mathbf{u}), g_2(\mathbf{u}), \cdots, g_m(\mathbf{u})]^\top)$. Then f is also a PL function (proof in Appendix B.1).*

Using the above proposition, we can reformulate the problem of optimizing the parameters of one layer of the network as a DC program. Specifically, the following proposition shows that the problem can be formulated as a latent structured SVM objective (Yu and Joachims, 2009).

Proposition 2. *The learning objective of a PL-CNN with respect to the parameters of the l-th layer can be specified as follows:*

$$\begin{aligned}\min_{W^l} \quad & \Big\{\frac{\lambda}{2}\|W^l\|_F^2 + \frac{1}{N}\sum_{i=1}^{N} \max_{\substack{\overline{\mathbf{h}}_i \in \mathcal{H} \\ \bar{y}_i \in \mathcal{Y}}} \Big(\Delta(\bar{y}_i, y_i) + (W^l)^\top \Psi(\mathbf{x}_i, \bar{y}_i, \overline{\mathbf{h}}_i)\Big) \\ & \qquad - \max_{\mathbf{h}_i \in \mathcal{H}} \Big((W^l)^\top \Psi(\mathbf{x}_i, y_i, \mathbf{h}_i)\Big)\Big\},\end{aligned} \tag{3.4}$$

for an appropriate choice of the latent space $\mathcal{H}$ and joint feature vectors $\Psi(\mathbf{x}, y, \mathbf{h})$ of the input $\mathbf{x}$, the output y and the latent variables $\mathbf{h}$. In other words, parameter estimation for the l-th layer corresponds to minimizing the sum of its Frobenius norm plus a PL function for each training sample.

Proof. For a given image $\mathbf{x}$ with the ground-truth class y, consider the input to the layer l, which we denote by $\mathbf{z}^{l-1}$. Since all the layers except the l-th one are fixed, the input $\mathbf{z}^{l-1}$ is a constant vector, which only depends on the image $\mathbf{x}$ (that is, its value does not depend on the variables W^l). In other words, we can write $\mathbf{z}^{l-1} = \varphi(\mathbf{x})$.

Given the input $\mathbf{z}^{l-1}$, all the elements of the output of the l-th layer, denoted by $\mathbf{z}^l$, are a PL function of W^l since the layer performs a linear transformation of $\mathbf{z}^{l-1}$ according to the parameters W^l, followed by an application of PL operations such as ReLU or max-pool. The vector $\mathbf{z}^l$ is then fed to the $(l+1)$-th layer. The output $\mathbf{z}^{l+1}$ of the $(l+1)$-th layer is a vector whose elements are PL functions of $\mathbf{z}^l$. Therefore, by proposition (1), the elements of $\mathbf{z}^{l+1}$ are a PL function of W^l. By applying the same argument until we reach the layer L, we can conclude that the representation $\Phi(\mathbf{x}; \mathcal{W})$ is a PL function of W^l.

Next, consider the upper bound of the empirical risk, which is specified as follows:

$$\max_{\bar{y} \in \mathcal{Y}} \left(\Delta(\bar{y}, y) + \left(W_{\bar{y}}^{\text{svm}} - W_y^{\text{svm}} \right)^T \Phi(\mathbf{x}; \mathcal{W}) \right). \tag{3.5}$$

Once again, since W^{svm} is fixed, the above upper bound can be interpreted as a PL function of $\Phi(\mathbf{x}; \mathcal{W})$, and thus, by proposition (1), the upper bound is a PL function of W^l. It only remains to observe that the learning objective (3.3) also contains the Frobenius norm of W^l. Thus, it follows that the estimation of the parameters of layer l can be reformulated as minimizing the sum of its Frobenius norm and the PL upper bound of the empirical risk over all training samples, as shown in problem (3.4). Note that we have ignored the constants corresponding to the Frobenius norm of the parameters of all the fixed layers. This constitutes an existential proof of Proposition 2. In the next paragraph, we give an intuition about the feature vectors $\Psi(\mathbf{x}_i, \bar{y}_i, \overline{\mathbf{h}}_i)$ and the latent space $\mathcal{H}$. □

Feature Vectors & Latent Space. The exact form of the joint feature vectors depends on the explicit DC decomposition of the objective function. In Appendix B.2, we detail the practical computations and give an example: we construct two

interleaved neural networks whose outputs define the convex and concave parts of the DC objective function. Given the explicit DC objective function, the feature vectors are given by a subgradient and can therefore be obtained by automatic differentiation.

We now give an intuition of what the latent space $\mathcal{H}$ represents. Consider an input image $\mathbf{x}$ and a corresponding latent variable $\mathbf{h} \in \mathcal{H}$. The latent variable can be viewed as a set of variables $\mathbf{h}^k, k \in \{l+1, \cdots, L\}$. In other words, each subset $\mathbf{h}^k$ of the latent variable corresponds to one of the layers of the network that follow the layer l. Intuitively, $\mathbf{h}^k$ represents the choice of activation at layer k when going through the PL activation: for each neuron j of layer k, $\mathbf{h}^k_j$ takes value i if and only if the i-th piece of the piecewise linear activation is selected. For instance, i is the index of the selected input in the case of a max-pooling unit.

Note that the latent space only depends on the layers that follow the current layer being optimized. This is due to the fact that the input $\mathbf{z}^{l-1}$ to the l-th layer is a constant vector that does not depend on the value of W^l. However, the activations of all subsequent layers following the l-th one depend on the value of the parameters W^l. As a consequence, the greater the number of following layers, the greater the size of the latent space, and this growth happens to be exponential. However, as will be seen shortly, it is still possible to efficiently optimize problem (3.4) for all the layers of the network despite this exponential increase.

3.4.2 Concave-Convex Procedure

The optimization problem (3.4) is a DC program in the parameters W^l. This follows from the fact that the upper bound of the empirical risk is a PL function, and can therefore be expressed as the difference of two convex PL functions (Melzer, 1986). Furthermore, the Frobenius norm of W^l is also a convex function of W^l. This observation allows us to obtain an approximate solution of problem (3.4) using the iterative concave-convex procedure (CCCP) (Yuille and Rangarajan, 2002).

Algorithm 3 describes the main steps of CCCP. In step 3, we impute the best value of the latent variable corresponding to the ground-truth class y_i for each

training sample. This imputation corresponds to the linearization step of the CCCP. The selected latent variable corresponds to a choice of activations at each non-linear layer of the network, and therefore defines a path of activations to the ground truth. Next, in step 4, we update the parameters by solving a convex optimization problem. This convex problem amounts to finding the path of activations which minimizes the maximum margin violations given the path to the ground truth defined in step 3.

The CCCP algorithm has the desirable property of providing a monotonic decrease in the objective function at each iteration. In other words, the objective function value of problem (3.4) at W_t^l is greater than or equal to its value at W_{t+1}^l. Since layerwise optimization itself can be viewed as a block-coordinate algorithm for minimizing the learning objective (3.3), our overall algorithm provides guarantees of monotonic decrease until convergence. This is one of the main advantages of our approach compared to backpropagation and its variants, which fail to provide similar guarantees on the value of the objective function from one iteration to the next.

Algorithm 3 *CCCP for parameter estimation of the l-th layer of the PL-CNN.*

Require: Data set $\mathcal{D} = \{(\mathbf{x}_i, y_i), i \in [N]\}$, fixed parameters $\{\mathcal{W} \cup W^{\text{svm}}\} \backslash W^l$, initial estimate W_0^l.

1: t = 0

2: **repeat**

3: For each sample $(\mathbf{x}_i, y_i)$, find the best latent variable value by solving the following problem:

$$\mathbf{h}_i^* = \underset{\overline{\mathbf{h}} \in \mathcal{H}}{\operatorname{argmax}} (W_t^l)^\top \Psi(\mathbf{x}_i, y_i, \overline{\mathbf{h}}). \tag{3.6}$$

4: Update the parameters by solving the following convex optimization problem:

$$W_{t+1}^l = \underset{W^l}{\operatorname{argmin}} \frac{\lambda}{2} \|W^l\|_F^2 + \frac{1}{N} \sum_{i=1}^{N} \max_{\substack{\bar{y}_i \in \mathcal{Y} \\ \overline{\mathbf{h}}_i \in \mathcal{H}}} \Big(\Delta(\bar{y}_i, y_i) + (W^l)^\top \Psi(\mathbf{x}_i, \bar{y}_i, \overline{\mathbf{h}}_i) \Big) - \Big((W^l)^\top \Psi(\mathbf{x}_i, y_i, \mathbf{h}_i^*) \Big). \tag{3.7}$$

5: t = t+1

6: **until** Objective function of problem (3.4) cannot be improved beyond a specified tolerance.

In order to solve the convex program (3.7), which corresponds to a structured SVM problem, we make use of the powerful BCFW algorithm (Lacoste-Julien and

Jaggi, 2013) that solves its dual via conditional gradients. This has two main advantages: (i) as the dual is a smooth quadratic program, each iteration of BCFW provides a monotonic increase in its objective; and (ii) the optimal step-size at each iteration can be computed analytically. This is once again in stark contrast to backpropagation, where the estimation of the step-size is still an active area of research (Duchi et al., 2011; Zeiler, 2012; Kingma and Ba, 2015). As shown by (Lacoste-Julien and Jaggi, 2013), given the current estimate of the parameters W^l, the conditional gradient of the dual of program (3.7) with respect to a training sample $(\mathbf{x}_i, y_i)$ can be obtained by solving the following problem:

$$(\hat{y}_i, \hat{\mathbf{h}}_i) = \underset{\bar{y}\in\mathcal{Y}, \overline{\mathbf{h}}\in\mathcal{H}}{\operatorname{argmax}} (W^l)^\top \Psi(\mathbf{x}_i, \bar{y}, \overline{\mathbf{h}}) + \Delta(\bar{y}, y_i). \tag{3.8}$$

We refer the interested reader to (Lacoste-Julien and Jaggi, 2013) for further details.

The overall efficiency of the CCCP algorithm relies on our ability to solve problems (3.6) and (3.8). At first glance, these problems may appear to be computationally intractable as the latent space $\mathcal{H}$ can be very large, especially for layers close to the input (of the order of millions of dimensions for a typical network). However, the following proposition shows that both the problems can be solved efficiently using the forward and backward passes that are employed in backpropagation.

Proposition 3. *Given the current estimate W^l of the parameters for the l-th layer, as well as the parameter values of all the other fixed layers, problems (3.6) and (3.8) can be solved using a forward pass on the network. Furthermore, the joint feature vectors $\Psi(\mathbf{x}_i, \hat{y}_i, \hat{\mathbf{h}}_i)$ and $\Psi(\mathbf{x}_i, y_i, \mathbf{h}_i^*)$ can be computed using a backward pass on the network.*

Proof. Recall that the latent space consists of the putative activations for each PL operation in the layers following the current one. Thus, intuitively, the maximization over the latent variables corresponds to finding the exact activations of all such PL operations. In other words, we need to identify the indices of the linear pieces that are used to compute the value of the PL function in the current state of the

network. For a ReLU operation, this corresponds to estimating $\max\{0, v\}$, where the input to the ReLU is a scalar v. Similarly, for a max-pool operation, this corresponds to estimating $\max_i\{\mathbf{e}_i^\top \mathbf{u}\}$, where $\mathbf{u}$ is the input vector to the max-pool. This is precisely the computation that the forward pass of backpropagation performs. Given the activations, the joint feature vector is the subgradient of the sample with respect to the current layer. Once again, this is precisely what is computed during the backward pass of the backpropagation algorithm. □

An example is constructed in Appendix B.2 to illustrate how to compute the feature vectors in practice.

3.4.3 Improving the BCFW Algorithm

As the BCFW algorithm was originally designed to solve a structured SVM problem, it requires further extensions to be suitable for training a PL-CNN. In what follows, we present three such extensions that improve the initialization, memory requirements and time complexity of the BCFW algorithm respectively.

Trust-Region for Initialization. The original BCFW algorithm starts with an initial parameter $W^l = \mathbf{0}$ (that is, all the parameters are set to 0). The reason for this initialization is that it is possible to compute the dual variables that correspond to the $\mathbf{0}$ primal variable. However, since our algorithm visits each layer of the network several times, it would be desirable to initialize its parameters using its current value W_l^t. To this end, we introduce a trust-region in the constraints of problem (3.7), or equivalently, an ℓ_2 norm based proximal term in its objective function (Parikh and Boyd, 2014). The following proposition shows that this has the desired effect of initializing the BCFW algorithm close to the current parameter values.

Proposition 4. *By adding a proximal term $\frac{\mu}{2}\|W^l - W_t^l\|_F^2$ to the objective function in (3.7), we can compute a feasible dual solution whose corresponding primal solution is equal to $\frac{\mu}{\lambda+\mu}W_t^l$. Furthermore, the addition of the proximal term still allows us to efficiently compute the conditional gradient using a forward-backward pass (proof in Appendix B.4).*

In practice, we always choose a value of $\mu = 10\lambda$: this yields an initialization of $\simeq 0.9W_t^l$ which does not significantly change the value of the objective function.

Efficient Representation of Joint Feature Vectors. The BCFW algorithm requires us to store a linear combination of the feature vectors for each mini-batch. While this requirement is not too stringent for convolutional and multi-class SVM layers, where the dimensionality of the feature vectors is small, it becomes prohibitively expensive for dense layers. The following proposition prevents a blow-up in the memory requirements of BCFW.

Proposition 5. *When optimizing dense layer l, if $W^l \in \mathbb{R}^{p\times q}$, we can store a representation of the joint feature vectors $\Psi(\mathbf{x}, y, \mathbf{h})$ with vectors of size p in problems (3.6) and (3.7). This is in contrast to the naïve approach that requires them to be of size $p \times q$.*

Proof. By Proposition (3), the feature vectors are subgradients of the hinge loss function, which we loosely denote by η for this proof. Then by the chain rule: $\frac{\partial \eta}{\partial W^l} = \frac{\partial \eta}{\partial z^l}\frac{\partial z^l}{\partial W^l} = \frac{\partial \eta}{\partial z^l} \cdot \left(z^{l-1}\right)^T$. Noting that $z^{l-1} \in \mathbb{R}^q$ is a forward pass up until layer l (independent of W^l), we can store only $\frac{\partial \eta}{\partial z^l} \in \mathbb{R}^p$ and still reconstruct the full feature vector $\frac{\partial \eta}{\partial W^l}$ by a forward pass and an outer product. □

Reducing the Number of Constraints. In order to reduce the amount of time required for the BCFW algorithm to converge, we use the structure of $\mathcal{H}$ to simplify problem (3.7) to a much simpler problem. Specifically, since $\mathcal{H}$ represents the activations of the network for a given sample, it has a natural decomposition over the layers: $\mathcal{H} = \mathcal{H}_1 \times ... \times \mathcal{H}_L$. We use this structure in the following observation.

Observation 1. *Problem (3.7) can be approximately solved by optimizing the dual problem on increasingly large search spaces. In other words, we start with constraints of $\mathcal{Y}$, followed by $Y \times \mathcal{H}_L$, then $\mathcal{Y} \times \mathcal{H}_L \times \mathcal{H}_{L-1}$ and so on. The algorithm converges when the primal-dual gap is below tolerance.*

The latent variables which are not optimized over are set to be the same as the ones selected for the ground truth. Experimentally, we observe that for convolutional layers (architectures in section 3.5), restricting the search space to $\mathcal{Y}$ yields a dual gap low enough to consider the problem has converged. This means that in practice for these layers, problem (3.7) can be solved by searching directions over the search space $\mathcal{Y}$ instead of the much larger $\mathcal{Y} \times \mathcal{H}$. The intuition is that the norm of the difference-of-convex decomposition grows with the number of activations selected differently in the convex and concave parts (see Appendix B.1 for the decomposition of piecewise linear functions). This compels the path of activations to be the same in the convex and the concave part to avoid large margin violations, especially for convolutional layers which are followed by numerous non-linearities at the max-pooling layers.

3.5 Experiments

Our experiments are designed to assess the ability of LW-SVM (Layer-Wise SVM, our method) and the SGD baselines to optimize problem (3.3). To compare LW-SVM with the state-of-the-art variants of backpropagation, we look at the training and testing accuracies as well as the training objective value. Unlike dropout, which effectively learns an ensemble model, we learn a single model using each baseline optimization algorithm. All experiments are conducted on a GPU (Nvidia Titan X) and use Theano (Bergstra et al., 2010; Bastien et al., 2012). We compare LW-SVM with Adagrad, Adadelta and Adam. For all data sets, we start at a good solution provided by these solvers and fine-tune it with LW-SVM. We then check whether a longer run of the SGD solver reaches the same level of performance.

The practical use of the LW-SVM algorithm needs choices at the three following levels: how to select the layer to optimize (i), when to stop the CCCP on each layer (ii) and when to stop the convex optimization at each inner iteration of the CCCP (iii). These choices are detailed in the next paragraph.

The layer-wise schedule of LW-SVM is as follows: as long as the validation accuracy increases, we perform passes from the end of the network (SVM) to the first layer (i). At each pass, each layer is optimized with one outer iteration of

the CCCP (ii). The inner iterations are stopped when the dual objective function does not increase by more than 1% over an epoch (iii). We point out that the dual objective function is cheap to compute since we are maintaining its value at all time. By contrast, to compute the exact primal objective function requires a forward pass over the data set without any update.

3.5.1 MNIST Data Set

Data set & Architecture The training data set consists in 60,000 gray scale images of size 28×28 with 10 classes, which we split into 50,000 samples for training and 10,000 for validating. The images are normalized, and we do not use any data augmentation. The architecture used for this experiment is shown in Figure 3.1.

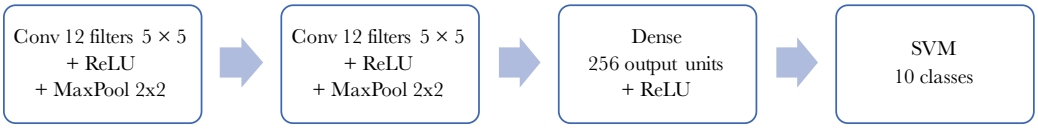


Figure 3.1: *Network architecture for the MNIST data set.*

Method The number of epochs is set to 200, 100 and 100 for Adagrad, Adadelta and Adam - Adagrad is given more epochs as we observed it took a longer time to converge. We then use LW-SVM and compare the results on training objective, training accuracy and testing accuracy. We also let the solvers run to up to 500 epochs to verify that we have not stopped the optimization prematurely. The regularization hyperparameter λ and the initial learning rate are chosen by cross-validation. λ is set to 0.001 for all solvers, and the initial learning rates can be found in Appendix B.3. For LW-SVM, λ is set to the same value as the baseline, and the proximal term μ to $\mu = 10\lambda = 0.01$.

Results As Table 3.1 shows, LW-SVM systematically improves on all training objective, training accuracy and testing accuracy. In particular, it obtains the best testing accuracy when combined with Adadelta. Because each convex sub-problem is run up to sufficient convergence, the objective function of LW-SVM

Table 3.1: *Results on MNIST: we compare the performance of LW-SVM with SGD algorithms on three metrics: training objective, training accuracy and testing accuracy. LW-SVM outperforms Adadelta and Adam on all three metrics, with marginal improvements since those find already very good solutions.*

Solver (epochs)	Training Objective	Training Accuracy	Time (s)	Testing Accuracy
Adagrad (200)	0.027	99.94%	707	99.22%
Adagrad (500)	0.024	99.96%	1759	99.20%
Adagrad (200) + LW-SVM	0.025	99.94%	707+366	99.21%
Adadelta (100)	0.049	99.56%	124	98.96%
Adadelta (500)	0.048	99.48%	619	99.05%
Adadelta (100) + LW-SVM	0.033	99.85%	124+183	**99.24%**
Adam (100)	0.038	99.76%	333	99.19%
Adam (500)	0.038	99.72%	1661	99.23%
Adam (100) + LW-SVM	0.029	99.89%	333+353	99.23%

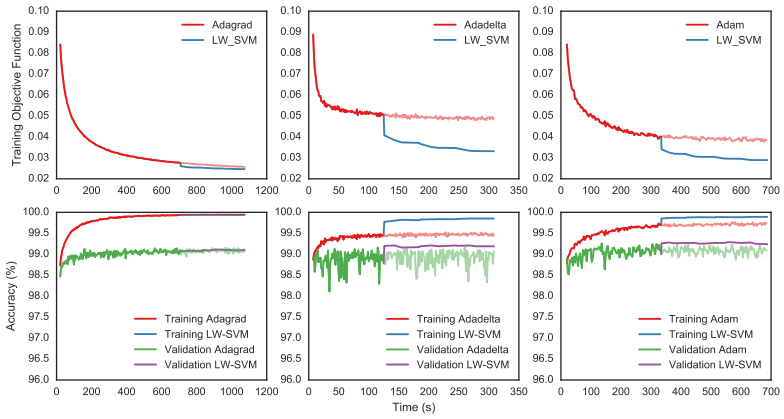


Figure 3.2: *Results on MNIST of Adagrad, Adadelta and Adam followed by LW-SVM. We verify that switching to LW-SVM leads to better solutions than running SGD longer (shaded continued plots).*

features of monotonic decrease at each iteration of the CCCP (blue curves in first row of Figure 3.2).

3.5.2 CIFAR Data Sets

Data sets & Architectures The CIFAR-10/100 data sets are comprised of 60,000 RGB natural images of size 32×32 with 10/100 classes (Krizhevsky, 2009)). We split the training set into 45,000 training samples and 5,000 validation samples in both cases. The images are centered and normalized, and we do not use any

data augmentation. To obtain a strong enough baseline, we employ (i) a pre-training with a softmax and cross-entropy loss and (ii) Batch-Normalization (BN) layers before each non-linearity.

We have experimentally found out that pre-training with a softmax layer followed by a cross-entropy loss led to better behavior and results than using an SVM loss alone. The baselines are trained with batch normalization. Once they have converged, the estimated mean and standard deviation are fixed like they would be at test time. Then batch normalization becomes a linear transformation, which can be handled by the LW-SVM algorithm. This allows us to compare LW-SVM with a baseline benefiting from batch normalization. Specifically, we use the architecture shown in Figure 3.3:

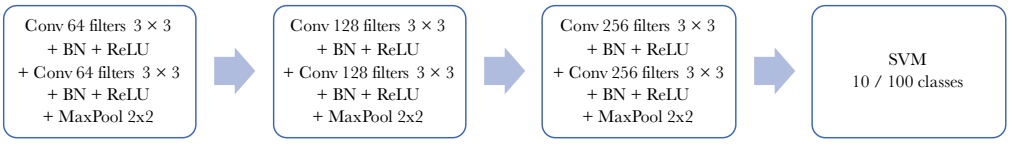


Figure 3.3: *Network architecture for the CIFAR data sets.*

Method Again, the initial learning rates and regularization weight λ are obtained by cross-validation, and a value of 0.001 is obtained for λ for all solvers on both data sets. As before, μ is set to 10λ. The initial learning rates are reported in Appendix B.3. The layer schedule and convergence criteria are as described at the beginning of the section. For each SGD optimizer, we train the network for 10 epochs with a cross-entropy loss (preceded by a softmax layer). Then it is trained with an SVM loss (without softmax) for respectively 1000, 100 and 100 epochs for Adagrad, Adadelta and Adam. This amount is doubled to verify that the baselines are not harmed by a premature stopping. Results are presented in Tables 3.2 and 3.3.

Table 3.2: *Results on CIFAR-10: LW-SVM outperforms Adam and Adadelta on all three metrics. It improves on Adagrad, but does not outperform it - however Adagrad takes a long time to converge and does not obtain the best generalization.*

Solver (epochs)	**Training Objective**	**Training Accuracy**	**Time (h)** (h)	**Testing Accuracy**
Adagrad (1000)	0.059	98.42%	10.58	83.15%
Adagrad (2000)	0.009	100.00%	21.14	83.84%
Adagrad (1000) + LW-SVM	0.012	100.00%	10.58+1.66	83.43%
Adadelta (100)	0.113	97.96%	0.83	84.42%
Adadelta (200)	0.054	99.83%	1.66	85.02%
Adadelta (100) + LW-SVM	0.038	100.00%	0.83+0.68	**86.62%**
Adam (100)	0.113	98.27%	0.83	84.18%
Adam (200)	0.055	99.76%	1.65	82.55%
Adam (100) + LW-SVM	0.034	100.00%	0.83+1.07	85.52%

Table 3.3: *Results on CIFAR-100: LW-SVM improves on all other solvers and obtains the best testing accuracy.*

Solver (epochs)	**Training Objective**	**Training Accuracy**	**Time (h)**	**Testing Accuracy**
Adagrad (1000)	0.201	95.36%	10.68	54.00%
Adagrad (2000)	0.044	99.98%	21.20	54.55%
Adagrad (1000) + LW-SVM	0.062	99.98%	10.68+3.40	53.97%
Adadelta (100)	0.204	95.68%	0.84	58.71%
Adadelta (200)	0.088	99.90%	1.67	58.03%
Adadelta (100) + LW-SVM	0.052	99.98%	0.84+1.48	**61.20%**
Adam (100)	0.221	95.79%	0.84	58.32%
Adam (200)	0.088	99.87%	1.66	57.81%
Adam (100) + LW-SVM	0.059	99.98%	0.84+1.69	60.17%

Results It can be seen from this set of results that LW-SVM *always* improves over the solution of the SGD algorithm, for example on CIFAR-100, decreasing the objective value of Adam from 0.22 to 0.06, or improving the test accuracy of Adadelta from 84.4% to 86.6% on CIFAR-10. The automatic step-size allows for a precise fine-tuning to optimize the training objective, while the regularization of the proximal term helps for better generalization.

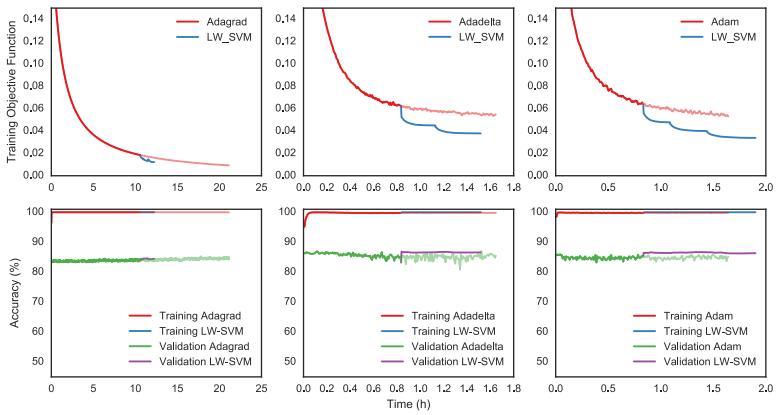


Figure 3.4: *Results on CIFAR-10 of Adagrad, Adadelta and Adam followed by LW-SVM. The successive drops of the training objective function with LW-SVM correspond to the passes over the layers.*

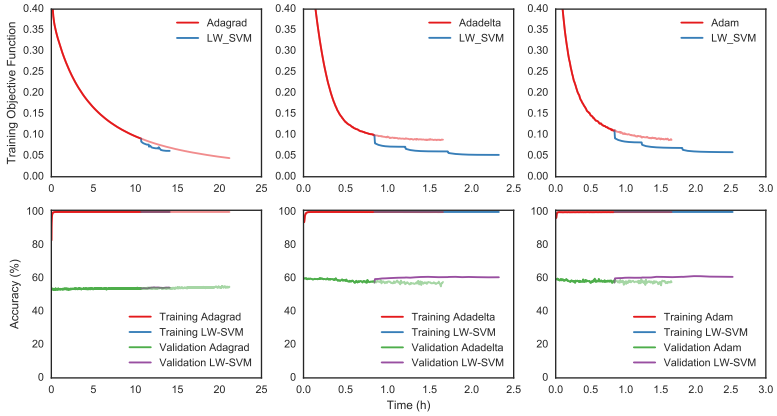


Figure 3.5: *Results on CIFAR-100 of Adagrad, Adadelta and Adam followed by LW-SVM. Although Adagrad keeps improving the training objective function, it takes much longer to converge and the improvement on the training and testing accuracies rapidly become marginal.*

3.5.3 ImageNet Data Set

We show results on the classification task of the ImageNet data set (Russakovsky et al., 2015). The ImageNet data set contains 1.2 million images for training and 50,000 images for validation, each of them mapped to one of the 1,000 classes. For this experiment we use a VGG-16 network (configuration D in (Simonyan and Zisserman, 2015)). We start with a pre-trained model as publicly available online, and we tune each of the dense layers as well as the final SVM layer with the LW-SVM algorithm. This experiment is designed to test the scalability of LW-SVM to large data sets and large networks, rather than comparing with the optimization

baselines as before - indeed for any baseline, obtaining proper convergence as in previous experiments would take a very long time. We set the hyperparameters λ to 0.001 and μ to 10λ as previously. We budget five epochs per layer, which in total takes two days of training on a single GPU (Nvidia Titan X). At training time we used centered crops of size 224×224. The evaluation method is the same as the single test scale method described in (Simonyan and Zisserman, 2015). We report the results on the validation set in Table 3.4, for the Pre-Trained model (PT) and the same model further optimized by LW-SVM (PT+LW-SVM):

Table 3.4: *Results on the 1,000-way classification challenge of ImageNet on the validation set, for the Pre-Trained model (PT) and the same model further optimized by LW-SVM (PT+LW-SVM).*

Network	**Top-1 Accuracy**	**Top-5 Accuracy**
VGG-16 (PT)	73.30%	91.33%
VGG-16 (PT + LW-SVM)	73.81%	91.61%

Since the objective function penalizes the top-1 error, it is logical to observe that the improvement is most important on the top-1 accuracy. Importantly, having an efficient representation of feature vectors proves to be essential for such large networks: for instance, in the optimization of the first fully connected layer with a batch-size of 100, the use of our representation lowers the memory requirements of the BCFW algorithm from 7,600GB to 20GB, which can then fit in the memory of a powerful computer.

3.6 Discussion

We presented a novel layerwise optimization algorithm for a large and useful class of convolutional neural networks, which we term PL-CNNs. Our key observation is that the optimization of the parameters of one layer of a PL-CNN is equivalent to solving a latent structured SVM problem. As the problem is a DC program, it naturally lends itself to the iterative CCCP approach, which optimizes a convex structured SVM objective at each iteration. This allows us to leverage the advancements made in structured SVM optimization over the past decade to design a computationally

feasible approach for learning PL-CNNs. Specifically, we use the BCFW algorithm and extend it to improve its initialization, memory requirements and time complexity. In particular, this allows our method to not require the tuning of any learning rate. Using the publicly available MNIST, CIFAR-10 and CIFAR-100 data sets, we show that our approach provides a boost for learning PL-CNNs over the state of the art backpropagation algorithms. Furthermore, we demonstrate scalability of the method with results on the ImageNet data set with a large network.

When the mean and standard deviation estimations of batch normalization are not fixed (unlike in our experiments with LW-SVM), batch normalization is not a piecewise linear transformation, and therefore cannot be used in conjunction with the BCFW algorithm for SVMs. However, it is difference-of-convex as it is a $\mathcal{C}^2$ function (Horst and Thoai, 1999). Incorporating a normalization scheme into our framework will be the object of future work. With our current methodology, LW-SVM algorithm can already be used on most standard architectures like VGG, Inception and ResNet-type architectures.

It is worth noting that other approaches for solving structured SVM problems, such as cutting-plane algorithms (Tsochantaridis et al., 2004; Joachims et al., 2009) Shalev-Shwartz2009 stochastic subgradient descent (Shalev-Shwartz et al., 2009), also rely on the efficiency of estimating the conditional gradient of the dual. Hence, all these methods are equally applicable to our setting. Indeed, the main strength of our approach is the establishment of a hitherto unknown connection between CNNs and latent structured SVMs. We believe that our observation will allow researchers to transfer the substantial existing knowledge of DC programs in general, and latent SVMs specifically, to produce the next generation of principled optimization algorithms for deep learning. In fact, there are already several such improvements that can be readily applied in our setting, which were not explored only due to a lack of time. This includes multi-plane variants of BCFW (Shah et al., 2015; Osokin et al., 2016), as well as generalizations of Frank-Wolfe such as partial linearization (Mohapatra et al., 2016).

Acknowledgments

This work was supported by the EPSRC AIMS CDT grant EP/L015987/1, the EPSRC Programme Grant Seebibyte EP/M013774/1 and Yougov. Many thanks to A. Desmaison, R. Bunel and D. Bouchacourt for the helpful discussions.

4

Deep Frank-Wolfe For Neural Network Optimization

Contents

Abstract

Learning a deep neural network requires solving a challenging optimization problem: it is a high-dimensional, non-convex and non-smooth minimization problem with a large number of terms. The current practice in neural network optimization is to rely on the stochastic gradient descent (SGD) algorithm or its adaptive variants. However, SGD requires a hand-designed schedule for the learning rate. In addition, its adaptive variants tend to produce solutions that generalize less well on unseen data than SGD with a hand-designed schedule. We present an optimization method that offers empirically the best of both worlds: our algorithm yields good generalization performance while requiring only one hyper-parameter. Our approach is based on a composite proximal framework, which exploits the compositional nature of deep neural networks and can leverage powerful convex optimization algorithms by design. Specifically, we employ the Frank-Wolfe (FW) algorithm for SVM, which computes an optimal step-size in closed-form at each time-step. We further show that the descent direction is given by a simple backward pass in the network, yielding the same computational cost per iteration as SGD. We present experiments on the CIFAR and SNLI data sets, where we demonstrate the significant superiority of our method over Adam, Adagrad, as well as the recently proposed BPGrad and AMSGrad. Furthermore, we compare our algorithm to SGD with a hand-designed learning rate schedule, and show that it provides similar generalization while often converging faster. The code is publicly available at `https://github.com/oval-group/dfw`.

4.1 Introduction

Since the introduction of back-propagation (Rumelhart et al., 1986), stochastic gradient descent (SGD) has been the most commonly used optimization algorithm for deep neural networks. While yielding remarkable performance on a variety of learning tasks, a downside of the SGD algorithm is that it requires a schedule for the decay of its learning rate. In the convex setting, curvature properties of the

objective function can be used to design schedules that are hyper-parameter free and guaranteed to converge to the optimal solution (Bubeck, 2015). However, there is no analogous result of practical interest for the non-convex optimization problem of a deep neural network. An illustration of this issue is the diversity of learning rate schedules used to train deep convolutional networks with SGD: (Simonyan and Zisserman, 2015) and (He et al., 2016) adapt the learning rate according to the validation performance, while (Szegedy et al., 2015; Huang et al., 2017) and (Loshchilov and Hutter, 2017) use pre-determined schedules, which are respectively piecewise constant, geometrically decaying, and cyclic with a cosine annealing. While these protocols result in competitive or state-of-the-art results on their learning task, there does not seem to be a consistent methodology. As a result, finding such a schedule for a new setting is a time-consuming and computationally expensive effort.

To alleviate this issue, adaptive gradient methods have been developed (Zeiler, 2012; Kingma and Ba, 2015; Reddi et al., 2018), and borrowed from online convex optimization (Duchi et al., 2011). Typically, these methods only require the tuning of the initial learning rate, the other hyper-parameters being considered robust across applications. However, it has been shown that such adaptive gradient methods obtain worse generalization than SGD (Wilson et al., 2017). This observation is corroborated by our experimental results.

In order to bridge this performance gap between existing adaptive methods and SGD, we introduce a new optimization algorithm, called Deep Frank-Wolfe (DFW). The DFW algorithm exploits the composite structure of deep neural networks to design an optimization algorithm that leverages efficient convex solvers. In more detail, we consider a composite (nested) optimization problem, with the loss as the outer function and the function encoded by the neural network as the inner one. At each iteration, we define a proximal problem with a first-order approximation of the neural network (linearized inner function), while keeping the loss function in its exact form (exact outer function). When the loss is the hinge loss, each proximal problem created by our formulation is exactly a linear SVM. This allows us to employ the powerful Frank-Wolfe (FW) algorithm as the workhorse of our procedure.

There are two by-design advantages to our method compared to the SGD algorithm. First, each iteration exploits more information about the learning objective, while preserving the same computational cost as SGD. Second, an optimal step-size is computed in closed-form by using the FW algorithm in the dual (Frank and Wolfe, 1956; Lacoste-Julien and Jaggi, 2013). Consequently, we do not need a hand-designed schedule for the learning rate. As a result, our algorithm is the first to provide competitive generalization error compared to SGD, all the while requiring a single hyper-parameter and often converging significantly faster.

We present two additional improvements to customize the use of the DFW algorithm to deep neural networks. First, we show how to smooth the loss function to avoid optimization difficulties arising from learning deep models with SVMs (Berrada et al., 2018). Second, we incorporate Nesterov momentum (Nesterov, 1983) to accelerate our algorithm.

We demonstrate the efficacy of our method on image classification with the CIFAR data sets (Krizhevsky, 2009) using two architectures: wide residual networks (Zagoruyko and Komodakis, 2016) and densely connected convolutional neural networks (Huang et al., 2017); we also provide experiments on natural language inference with a Bi-LSTM on the SNLI corpus (Bowman et al., 2015). We show that the DFW algorithm often strongly outperforms previous methods based on adaptive learning rates. Furthermore, it provides comparable or better accuracy to SGD with hand-designed learning rate schedules.

In conclusion, our contributions can be summed up as follows:

- We propose a proximal framework which preserves information from the loss function.
- For the first time for deep neural networks, we demonstrate how our formulation gives at each iteration (i) an optimal step-size in closed form and (ii) an update at the same computational cost as SGD.
- We design a novel smoothing scheme for the dual optimization of SVMs.

- To the best of our knowledge, the resulting DFW algorithm is the first to offer comparable or better generalization to SGD with a hand-designed schedule on the CIFAR data sets, all the while converging several times faster and requiring only a single hyperparameter.

4.2 Related Work

Non Gradient-Based Methods. The success of a simple first-order method such as SGD has led to research in other more sophisticated techniques based on relaxations (Heinemann et al., 2016; Zhang et al., 2017b), learning theory (Goel et al., 2017), Bregman iterations (Taylor et al., 2016), and even second-order methods (Roux et al., 2008; Martens and Sutskever, 2012; Ollivier, 2013; Desjardins et al., 2015; Martens and Grosse, 2015; Grosse and Martens, 2016; Ba et al., 2017; Botev et al., 2017; Martens et al., 2018). While such methods hold a lot of promise, their relatively large per-iteration cost limits their scalability in practice. As a result, gradient-based methods continue to be the most popular optimization algorithms for learning deep neural networks.

Adaptive Gradient Methods. As mentioned earlier, one of the main challenges of using SGD is the design of a learning rate schedule. Several works proposed alternative first-order methods that do not require such a schedule, by either modifying the descent direction or adaptively rescaling the step-size (Duchi et al., 2011; Zeiler, 2012; Schaul et al., 2013; Kingma and Ba, 2015; Zhang et al., 2017c; Reddi et al., 2018). However, as noted above, the adaptive variants of SGD sometimes provide subpar generalization (Wilson et al., 2017).

Learning to Learn and Meta-Learning. Learning to learn approaches have also been proposed to optimize deep neural networks. (Baydin et al., 2018) and (Wu et al., 2018) learn the learning rate to avoid a hand-designed schedule and to improve practical performance. Such methods can be combined with our proposed

algorithm to learn its proximal coefficient, instead of considering it as a fixed hyper-parameter to be tuned. Meta-learning approaches have also been suggested to learn the optimization algorithm (Andrychowicz et al., 2016; Ravi and Larochelle, 2017; Wichrowska et al., 2017; Li and Malik, 2017). This line of work, which is orthogonal to ours, could benefit from the use of DFW to optimize the meta-learner.

Optimization and Generalization. Several works study the relationship between optimization and generalization in deep learning. In order to promote generalization within the optimization algorithm itself, (Neyshabur et al., 2015, 2016) proposed the Path-SGD algorithm, which implicitly controls the capacity of the model. However, their method required the model to employ ReLU non-linearity only, which is an important restriction for practical purposes. (Hardt et al., 2016; Arpit et al., 2017; Neyshabur et al., 2017; Hoffer et al., 2017) and (Chaudhari and Soatto, 2018) analyzed how existing optimization algorithms implicitly regularize deep neural networks. However this phenomenon is not yet fully understood, and the resulting empirical recommendations are sometimes opposing (Hardt et al., 2016; Hoffer et al., 2017).

Proximal Methods. The back-propagation algorithm has been analyzed in a proximal framework in (Frerix et al., 2018). Yet, the resulting approach still requires the same hyper-parameters as SGD and incurs a higher computational cost per iteration.

Linear SVM Sub-Problems. A main component of our formulation is to formulate sub-problems as linear SVMs. In an earlier work (Berrada et al., 2017), we showed that neural networks with piecewise linear activations could be trained with the CCCP algorithm (Yuille and Rangarajan, 2002), which yielded approximate SVM problems to be solved with the BCFW algorithm (Lacoste-Julien and Jaggi, 2013). However this algorithm only updates the parameters of one layer at a time, which slows down convergence significantly in practice. Closest to our approach are the works of (Hochreiter and Obermayer, 2005) and (Singh and Shawe-Taylor,

2018). (Hochreiter and Obermayer, 2005) suggested to create a local SVM based on a first-order Taylor expansion and a proximal term, in order to lower the error of every data sample while minimizing the changes in the weights. However their method operated in a non-stochastic setting, making the approach infeasible for large-scale data sets. (Singh and Shawe-Taylor, 2018), a parallel work to ours, also created an SVM problem using a first-order Taylor expansion, this time in a mini-batch setting. Their work provided interesting insights from a statistical learning theory perspective. While their method is well-grounded, its significantly higher cost per iteration impairs its practical speed and scalability. As such, it can be seen as complementary to our empirical work, which exploits a powerful solver and provides state-of-the-art scalability and performance.

4.3 Problem Formulation

Before describing our formulation, we introduce some necessary notation. We use $\|\cdot\|$ to denote the Euclidean norm. Given a function ϕ, $\partial\phi(\boldsymbol{u})\Big|_{\hat{\boldsymbol{u}}}$ is the derivative of ϕ with respect to $\boldsymbol{u}$ evaluated at $\hat{\boldsymbol{u}}$. According to the situation, this derivative can be a gradient, a Jacobian or even a directional derivative. Its exact nature will be clear from context throughout the paper. We also introduce the first-order Taylor expansion of ϕ around the point $\hat{\boldsymbol{u}}$: $\mathcal{T}_{\hat{\boldsymbol{u}}}\phi(\boldsymbol{u}) = \phi(\hat{\boldsymbol{u}}) + (\partial\phi(\boldsymbol{u})\Big|_{\hat{\boldsymbol{u}}})^\top(\boldsymbol{u} - \hat{\boldsymbol{u}})$. For a positive integer p, we denote the set $\{1, 2, ..., p\}$ as $[p]$. For simplicity, we assume that stochastic algorithms process only one sample at each iteration, although the methods can be trivially extended to mini-batches of size larger than one.

4.3.1 Learning Objective

We suppose we are given a data set $(\boldsymbol{x}_i, y_i)_{i\in[N]}$, where each $\boldsymbol{x}_i \in \mathbb{R}^d$ is a sample annotated with a label y_i from the output space $\mathcal{Y}$. The data set is used to estimate a parameterized model represented by the function $\boldsymbol{f}$. Given its (flattened) parameters $\boldsymbol{w} \in \mathbb{R}^p$, and an input $\boldsymbol{x}_i \in \mathbb{R}^d$, the model predicts $\boldsymbol{f}(\boldsymbol{w}, \boldsymbol{x}_i) \in \mathbb{R}^{|\mathcal{Y}|}$, a vector with one score per element of the output space $\mathcal{Y}$. For instance, $\boldsymbol{f}$ can be a linear map or a deep neural network. Given a vector of scores per label

$\boldsymbol{s} \in \mathbb{R}^{|\mathcal{Y}|}$, we denote by $\mathcal{L}(\boldsymbol{s}, y_i)$ the loss function that computes the risk of the prediction scores $\boldsymbol{s}$ given the ground truth label y_i. For example, the loss $\mathcal{L}$ can be cross-entropy or the multi-class hinge loss:

$$\text{(Cross-Entropy)} \quad \mathcal{L}_{CE} : (\boldsymbol{s}, y) \in \mathbb{R}^{|\mathcal{Y}|} \times \mathcal{Y} \mapsto \log\left(\sum_{k \in \mathcal{Y}} \exp(s_k)\right) - s_y, \tag{4.1}$$

$$\text{(Multi-Class Hinge)} \quad \mathcal{L}_{hinge} : (\boldsymbol{s}, y) \in \mathbb{R}^{|\mathcal{Y}|} \times \mathcal{Y} \mapsto \max\left\{\max_{k \in \mathcal{Y} \setminus \{y\}} \{s_k + 1 - s_y\}, 0\right\}. \tag{4.2}$$

The cross-entropy loss (4.1) tries to match the empirical distribution by driving incorrect scores as far as possible from the ground truth one. The hinge loss (4.2) attempts to create a minimal margin of one between correct and incorrect scores. The hinge loss has been shown to be more robust to over-fitting than cross-entropy, when combined with smoothing techniques that are common in the optimization literature (Berrada et al., 2018). To simplify notation, we introduce $\boldsymbol{f}_i(\boldsymbol{w}) = \boldsymbol{f}(\boldsymbol{w}, \boldsymbol{x}_i)$ and $\mathcal{L}_i(\boldsymbol{s}) = \mathcal{L}(\boldsymbol{s}, y_i)$ for each $i \in [N]$. Finally, we denote by $\rho(\boldsymbol{w})$ the regularization (typically the squared Euclidean norm). We now write the learning problem under its empirical risk minimization form:

$$\min_{\boldsymbol{w} \in \mathbb{R}^p} \rho(\boldsymbol{w}) + \frac{1}{N} \sum_{i \in [N]} \mathcal{L}_i(\boldsymbol{f}_i(\boldsymbol{w})). \tag{4.3}$$

4.3.2 A Proximal Approach

Our main contribution is a formulation which exploits the composite nature of deep neural networks in order to obtain a better approximation of the objective at each iteration. Thanks to the careful approximation design, this approach yields sub-problems that are amenable to efficient optimization by powerful convex solvers. In order to understand the intuition of our approach, we first present a proximal gradient perspective on SGD.

The SGD Algorithm. At iteration t, the SGD algorithm selects a sample j at random and observes the objective estimate $\rho(\boldsymbol{w}_t) + \mathcal{L}_j(\boldsymbol{f}_j(\boldsymbol{w}_t))$. Then, given the learning rate η_t, it performs the following update on the parameters:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \left(\partial \rho(\boldsymbol{w}) \Big|_{\boldsymbol{w}_t} + \partial \mathcal{L}_j(\boldsymbol{f}_j(\boldsymbol{w})) \Big|_{\boldsymbol{w}_t} \right). \tag{4.4}$$

Equation (4.4) is the closed-form solution of a proximal problem where the objective has been linearized by the first-order Taylor expansion $\mathcal{T}_{\boldsymbol{w}_t}$ (Bubeck, 2015):

$$\boldsymbol{w}_{t+1} = \underset{\boldsymbol{w} \in \mathbb{R}^p}{\operatorname{argmin}} \left\{ \frac{1}{2\eta_t} \|\boldsymbol{w} - \boldsymbol{w}_t\|^2 + \mathcal{T}_{\boldsymbol{w}_t} \rho(\boldsymbol{w}) + \mathcal{T}_{\boldsymbol{w}_t} [\mathcal{L}_j(\boldsymbol{f}_j(\boldsymbol{w}))] \right\}. \tag{4.5}$$

To see the relationship between (4.4) and (4.5), one can set the gradient with respect to $\boldsymbol{w}$ to 0 in equation (4.5), and observe that the resulting equation is exactly (4.4). In other words, SGD minimizes a first-order approximation of the objective, while encouraging proximity to the current estimate $\boldsymbol{w}_t$.

However, one can also choose to linearize only a part of the composite objective (Lewis and Wright, 2016). Choosing which part to approximate is a crucial decision, because it yields optimization problems with widely different properties. In this work, we suggest an approach that lends itself to fast optimization with robust convex solvers and preserves information about the learning task by keeping an exact loss function.

Loss-Preserving Linearization. In detail, at iteration t, with selected sample j, we introduce the proximal problem that linearizes the regularization ρ and the model $\boldsymbol{f}_j$, but not the loss function $\mathcal{L}$:

$$\min_{\boldsymbol{w} \in \mathbb{R}^p} \left\{ \frac{1}{2\eta_t} \|\boldsymbol{w} - \boldsymbol{w}_t\|^2 + \mathcal{T}_{\boldsymbol{w}_t} \rho(\boldsymbol{w}) + \mathcal{L}_j(\mathcal{T}_{\boldsymbol{w}_t} \boldsymbol{f}_j(\boldsymbol{w})) \right\}. \tag{4.6}$$

In figure 4.1, we provide a visual comparison of equations (4.5) and (4.6) in the case of a piecewise linear loss. As will be seen, by preserving the loss function, we will be able to achieve good performance across a number of tasks with a fixed $\eta_t = \eta$. Consequently, we will provide the first algorithm to accurately learn deep neural networks with only a single hyper-parameter while offering similar performance compared to SGD with a hand-designed schedule.

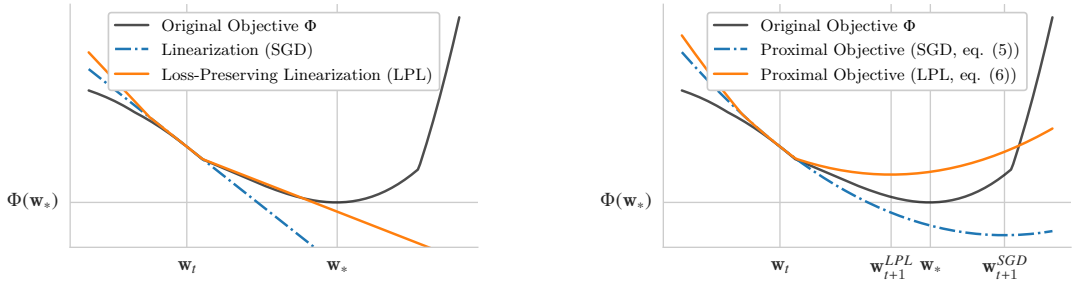


Figure 4.1: *We illustrate the different approximations on a synthetic composite objective function* $\Phi(\boldsymbol{w}) = \mathcal{L}(\boldsymbol{f}(\boldsymbol{w}))$ *(*Φ *is plotted in black). In this example,* $\mathcal{L}$ *is a maximum of linear functions (similarly to a hinge loss) and* $\boldsymbol{f}$ *is a non-linear smooth map. We denote the current iterate by* $\boldsymbol{w}_t$*, and the point minimizing* Φ *by* $\boldsymbol{w}_*$*. On the left-hand side, one can observe how the SGD approximation is a single line (tangent at* $\Phi(\boldsymbol{w}_t)$*, in blue), while the LPL approximation is piecewise linear (in orange), and thus matches the objective curve (in black) more closely. On the right-hand side, an identical proximal term is added to both approximations to visualize equations (4.5) and (4.6). Thanks to the better accuracy of the LPL approximation, the iterate* $\boldsymbol{w}_{t+1}^{LPL}$ *gets closer to the solution* $\boldsymbol{w}_*$ *than* $\boldsymbol{w}_{t+1}^{SGD}$*. This effect is particularly true when the proximal coefficient* $\frac{1}{2\eta_t}$ *is small, or equivalently, when the learning rate* η_t *is large. Indeed, the accuracy of the local approximation becomes more important when the proximal term is contributing less (e.g. when* η_t *is large).*

4.4 The Deep Frank-Wolfe Algorithm

4.4.1 Algorithm

We focus on the optimization of equation (4.6) when $\mathcal{L}$ is a multi-class hinge loss (4.2). The results of this section were originally derived for linear models (Lacoste-Julien and Jaggi, 2013). Our contribution is to show for the first time how they can be exploited for deep neural networks thanks to our formulation (4.6). We will refer to the resulting algorithm for neural networks as Deep Frank-Wolfe (DFW). We begin by stating the key advantage of our method.

Proposition 6 (Optimal step-size, (Lacoste-Julien and Jaggi, 2013))**.** *Problem (4.6) with a hinge loss is amenable to optimization with Frank-Wolfe in the dual, which yields an optimal step-size* $\gamma_t \in [0, 1]$ *in closed-form at each iteration* t*.*

This optimal step-size can be obtained in closed-form because the hinge loss is convex and piecewise linear. In fact, the approach presented here can be applied

to any loss function $\mathcal{L}$ that is convex and piecewise linear (another example would be the l_1 distance for regression for instance).

Since the step-size can be computed in closed-form, the main computational challenge is to obtain the update direction, that is, the conditional gradient of the dual. In the following result, we show that by taking a single step per proximal problem, this dual conditional gradient can be computed at the same cost as a standard stochastic gradient. The proof is available in appendix C.1.5.

Proposition 7 (Cost per iteration)**.** *If a single step is performed on the dual of (4.6), its conditional gradient is given by* $-\partial \left(\rho(\boldsymbol{w}) + \mathcal{L}_y(\boldsymbol{f}_{\boldsymbol{x}}(\boldsymbol{w}))\right)\Big|_{\boldsymbol{w}_t}$*. Given the step-size* γ_t*, the resulting update can be written as:*

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \left[\partial \rho(\boldsymbol{w})\Big|_{\boldsymbol{w}_t} + \gamma_t \partial \mathcal{L}_j(\boldsymbol{f}_j(\boldsymbol{w}))\Big|_{\boldsymbol{w}_t}\right] \tag{4.7}$$

In other words, the cost per iteration of the DFW algorithm is the same as SGD, since the update only requires standard stochastic gradients. In addition, we point out that in a mini-batch setting, the conditional gradient is given by the average of the gradients over the mini-batch. As a consequence, we can use batch Frank-Wolfe in the dual rather than coordinate-wise updates, with the same parallelism as SGD over the samples of a mini-batch.

One can observe how the update (4.7) exploits the optimal step-size $\gamma_t \in [0, 1]$ given by Proposition 6. There is a geometric interpretation to the role of this step-size γ_t. When γ_t is set to its minimal value 0, the resulting iterate does not move along the direction $\partial \mathcal{L}_j(\boldsymbol{f}_j(\boldsymbol{w}))\Big|_{\boldsymbol{w}_t}$. Since the step-size is optimal, this can only happen if the current iterate is detected to be at a minimum of the piecewise linear approximation. Conversely, when γ_t reaches its maximal value 1, the algorithm tries to move as far as possible along the direction $\partial \mathcal{L}_j(\boldsymbol{f}_j(\boldsymbol{w}))\Big|_{\boldsymbol{w}_t}$. In that case, the update is the same as the one obtained by SGD (as given by equation (4.4)). In other words, γ_t can automatically decay the effective learning rate, hereby preventing the need to design a learning rate schedule by hand.

As mentioned previously, the DFW algorithm performs only one step per proximal problem. Since problem (4.6) is only an approximation of the original

problem (4.3), it may be unnecessarily expensive to solve it very accurately. Therefore taking a single step per proximal problem may help the DFW algorithm to converge faster. This is confirmed by our experimental results, which show that DFW is often able to minimize the learning objective (4.3) at greater speed than SGD.

4.4.2 Improvements for Deep Neural Networks

We present two improvements to customize the application of our algorithm to deep neural networks.

Smoothing. The SVM loss is non-smooth and has sparse derivatives, which can cause difficulties when training a deep neural network (Berrada et al., 2018). In Appendix C.1.6, we derive a novel result that shows how we can exploit the smooth primal cross-entropy direction and inexpensively detect when to switch back to using the standard conditional gradient.

Nesterov Momentum. To take advantage of acceleration similarly to the SGD baseline, we adapt the Nesterov momentum to the DFW algorithm. We defer the details to the appendix in C.1.7 for space reasons. We further note that the momentum coefficient μ is typically set to a high value, say 0.9, and does not contribute significantly to the computational cost of cross-validation.

4.4.3 Algorithm Summary

The main steps of DFW are shown in Algorithm 4. As the key feature of our approach, note that the step-size is computed in closed-form in step 10 of the algorithm (colored in blue).

Note that only the hyper-parameter η will be tuned in our experiments: we will use the same batch-size, momentum and number of epochs as the baselines in our experiments (unless specified otherwise). In addition, we point out again that when $\gamma_t = 1$, we recover the SGD step with Nesterov momentum.

In sections C.1.5 and C.1.6 of the appendix, we detail the derivation of the optimal step-size (step 10) and the computation of the search direction (step 7).

Algorithm 4 *The Deep Frank-Wolfe Algorithm*

Require: proximal coefficient η, initial point $\boldsymbol{w}_0 \in \mathbb{R}^p$, momentum coefficient μ, number of epochs

$t = 0$
$\mathbf{z}_0 = 0$ ▷ Momentum velocity (initialization)
for each epoch **do**
  for each mini-batch $\mathcal{B}$ **do**
    Receive data of mini-batch $(\boldsymbol{x}_i, y_i)_{i\in\mathcal{B}}$
    $\forall i \in \mathcal{B},\ \mathbf{b}_t^{(i)}(\boldsymbol{w}_t) = (f_{\boldsymbol{x}_i,\bar{y}}(\boldsymbol{w}_t) - f_{\boldsymbol{x}_i,y_i}(\boldsymbol{w}_t) + \Delta(\bar{y}, y_i))_{\bar{y}\in\mathcal{Y}}$ ▷ Forward pass
    $\forall i \in \mathcal{B},\ \boldsymbol{s}_t^{(i)} \leftarrow \texttt{get\_s}(\mathbf{b}_t^{(i)}(\boldsymbol{w}_t))$ ▷ Dual direction (details in Appendix C.1.6)
    $\boldsymbol{\delta}_t = \partial\left(\frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}}(\boldsymbol{s}_t^{(i)})^\top \mathbf{b}_t^{(i)}(\boldsymbol{w})\right)\Big|_{\boldsymbol{w}_t}$ ▷ Derivative of (smoothed) loss function
    $\mathrm{r}_t = \partial\rho(\boldsymbol{w})\Big|_{\boldsymbol{w}_t}$ ▷ Derivative of regularization
    $\gamma_t = (-\eta\boldsymbol{\delta}_t^\top \mathrm{r}_t + \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}}(\boldsymbol{s}_t^{(i)})^\top \mathbf{b}_t^{(i)}(\boldsymbol{w}_t)/(\eta\|\boldsymbol{\delta}_t\|^2)$ clipped to $[0, 1]$ ▷ Step-size
    $\mathbf{z}_{t+1} = \mu\mathbf{z}_t - \eta\gamma_t(\mathrm{r}_t + \boldsymbol{\delta}_t)$ ▷ Velocity accumulation
    $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta\left[\mathrm{r}_t + \gamma_t\boldsymbol{\delta}_t\right] + \mu\mathbf{z}_{t+1}$ ▷ Parameters update
    $t = t + 1$
  end for
end for

The computation of the dual search direction is omitted here for space reasons. However, its implementation is straightforward in practice, and its computational cost is linear in the size of the output space.

Finally, we emphasize that the DFW algorithm is motivated by an empirical perspective. While our method is not guaranteed to converge, our experiments show an effective minimization of the learning objective for the problems encountered in practice.

4.5 Experiments

We compare the Deep Frank Wolfe (DFW) algorithm to the state-of-the-art optimizers. We show that, across diverse data sets and architectures, the DFW algorithm outperforms adaptive gradient methods (with the exception of one setting, DN-10, where it obtains similar performance to AMSGrad and BPGrad). In addition, the DFW algorithm offers competitive and sometimes superior performance to SGD

at a lower computational cost, even though SGD has the advantage of a hand-designed schedule that has been chosen separately for each of these tasks.

Our experiments are implemented in PyTorch (Paszke et al., 2017), and the code is available at `https://github.com/oval-group/dfw`. All models are trained on a single Nvidia Titan Xp card.

4.5.1 Image Classification with Convolutional Neural Networks

Data Set & Architectures. The CIFAR-10/100 data sets contain 60,000 RGB natural images of size 32×32 with 10/100 classes (Krizhevsky, 2009). We split the training set into 45,000 training samples and 5,000 validation samples, and use 10,000 samples for testing. The images are centered and normalized per channel. Unless specified otherwise, no data augmentation is employed. We perform our experiments on two modern architectures of deep convolutional neural networks: wide residual networks (Zagoruyko and Komodakis, 2016), and densely connected convolutional networks (Huang et al., 2017). Specifically, we employ a wide residual network of depth 40 and width factor 4, which has 8.9M parameters, and a "bottleneck" densely connected convolutional neural network of depth 40 and growth factor 40, which has 1.9M parameters. We refer to these architectures as WRN and DN respectively. All the following experimental details follow the protocol of (Zagoruyko and Komodakis, 2016) and (Huang et al., 2017). The only difference is that, instead of using 50,000 samples for training, we use 45,000 samples for training, and 5,000 samples for the validation set, which we found to be essential for all adaptive methods. While Deep Frank Wolfe (DFW) uses an SVM loss, the baselines are trained with the Cross-Entropy (CE) loss since this resulted in better performance.

Method. We compare DFW to the most common adaptive learning rates currently used: Adagrad (Duchi et al., 2011), Adam (Kingma and Ba, 2015), the corrected version of Adam called AMSGrad (Reddi et al., 2018), and BPGrad (Zhang et al., 2017c). For these methods and for DFW, we cross-validate the initial learning rate as a power of 10. We also evaluate the performance of SGD with momentum (simply

referred to as SGD), for which we follow the protocol of (Zagoruyko and Komodakis, 2016) and (Huang et al., 2017). For all methods, we set a budget of 200 epochs for WRN and 300 epochs for DN. Furthermore, the batch-size is respectively set to 128 and 64 for WRN and DN as in (Zagoruyko and Komodakis, 2016) and (Huang et al., 2017). For DN, the l_2 regularization is set to 10^{-4} as in (Huang et al., 2017). For WRN, the l_2 is cross-validated between 5.10^{-4}, as in (Zagoruyko and Komodakis, 2016), and 10^{-4}, a more usual value that we have found to perform better for some of the methods (in particular DFW, since the corresponding loss function is an SVM instead of CE, for which the value of 5.10^{-4} was designed). The value of the Nesterov momentum is set to 0.9 for BPGrad, SGD and DFW. DFW has only one hyper-parameter to tune, namely η, which is analogous to an initial learning rate. For SGD, the initial learning rate is set to 0.1 on both WRN and DN. Following (Zagoruyko and Komodakis, 2016) and (Huang et al., 2017), it is then divided by 5 at epochs 60, 120 and 180 for WRN, and by 10 at epochs 150 and 225 for DN.

Results. We present the results in Table 4.1.

Architecture	Optimizer	CIFAR-10 Test Accuracy (%)	CIFAR-100 Test Accuracy (%)
WRN	Adagrad	86.07	57.64
	Adam	84.86	58.46
	AMSGrad	86.08	60.73
	BPGrad	88.62	60.31
	DFW	**90.18**	**67.83**
	SGD	90.08	66.78
DN	Adagrad	87.32	56.47
	Adam	88.44	64.61
	AMSGrad	90.53	68.32
	BPGrad	**90.85**	59.36
	DFW	90.22	**69.55**
	SGD	**92.02**	**70.33**

Table 4.1: *Results on the CIFAR data sets without data augmentation. In black, all adaptive methods have a single hyper-parameter for their step-size. In red, SGD benefits from a hand-designed schedule. DFW outperforms all baselines on the WRN architecture, by a margin of 7% for adaptive gradient methods on CIFAR-100. On the DN-100 task, DFW exceeds the accuracy of Adagrad by 14%.*

Observe that DFW significantly outperforms the adaptive gradient methods, particularly on the more challenging CIFAR-100 data set. On the WRN-CIFAR-100 task in particular, DFW obtains a testing accuracy which is about 7% higher than all other adaptive methods and outperforms SGD with a hand-designed schedule by 1%. The inferior generalization of adaptive gradient methods is consistent with the findings of (Wilson et al., 2017). On all tasks, the accuracy of DFW is comparable to SGD. Note that DFW converges significantly faster than SGD: the network reaches its final performance several times faster than SGD in all cases. We illustrate this with an example in figure 4.2, which plots the training and validation errors on DN-CIFAR-100. In figure 4.3, one can see how the step-size is automatically decayed by DFW on this same experiment: we compare the effective step-size $\gamma_t \eta$ for DFW to the manually tuned η_t for SGD.

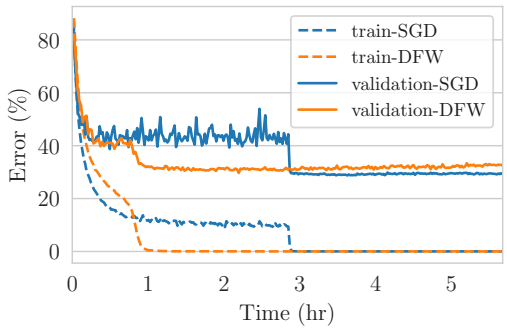


Figure 4.2: *Training and validation error during the training of DN on CIFAR-100. DFW converges significantly faster than SGD.*

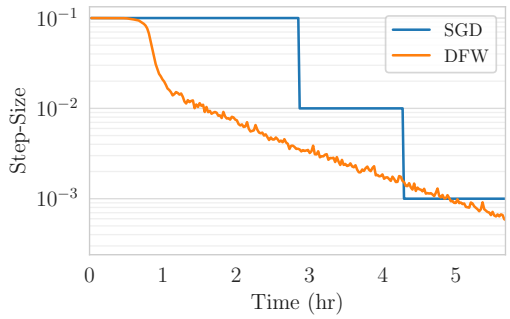


Figure 4.3: *The (automatic) evolution of $\gamma_t \eta$ for the DFW algorithm compared to the "staircase" hand-designed schedule of η_t for SGD.*

Data Augmentation. Since data augmentation provides a significant boost to the final accuracy, we provide additional results that make use of it. Specifically, we randomly flip the images horizontally and randomly crop them with four pixels padding. For methods that do not use a hand-designed schedule, such data augmentation introduces additional variance which makes the adaptation of the step-size more difficult. Therefore we allow the batch size of adaptive methods (e.g. all methods but SGD) to be chosen as 1x, 2x or 4x, where x is

the original value of batch-size (64 for DN, 128 for WRN). Due to the heavy computational cost of the cross-validation (we tune the batch-size, regularization and initial learning rate), we provide results for SGD, DFW and the best performing adaptive gradient method, which is AMSGrad. For SGD the hyper-parameters are kept the same as in (Zagoruyko and Komodakis, 2016) and (Huang et al., 2017). We present the results in Table 4.2.

Architecture	Optimizer	CIFAR-10 Test Accuracy (%)	CIFAR-100 Test Accuracy (%)
WRN	AMSGrad	90.06	67.75
	DFW	**94.71**	**74.71**
	SGD	**95.40**	**77.78**
	SGD*	**95.47**	**78.82**
DN	AMSGrad	91.78	69.58
	DFW	**94.88**	**73.20**
	SGD	**95.26**	**76.26**

Table 4.2: *Results on the CIFAR data sets with data augmentation. In black, all adaptive methods have a single hyper-parameter for their step-size. In red, SGD benefits from a hand-designed schedule. On the fourth line, SGD* refers to the result reported in Table 5 of (Huang et al., 2017). The small difference between the results of SGD and SGD* can be explained by the fact that we use 5,000 fewer training samples in our experiments (these are kept for validation). The results of this table show that DFW systematically outperforms AMSGrad on this task (by up to 7% on WRN-100).*

These results confirm that DFW consistently outperforms AMSGrad, which is the best adaptive baseline on these tasks. In particular, DFW obtains a test accuracy which is 7% better than AMSGrad on WRN-100.

4.5.2 Natural Language Inference with Recurrent Neural Networks

Data Set. The Stanford Natural Language Inference (SNLI) data set is a large corpus of 570k pairs of sentences (Bowman et al., 2015). Each sentence is labeled by one of the three possible labels: entailment, neutral and contradiction. This allows the model to learn the semantics of the text data from a three-way classification problem. Thanks to its scale and its supervised labels, this data set allows large neural networks to learn high-quality text embeddings. As (Conneau et al., 2017)

demonstrate, the SNLI corpus can thus be used as a basis for transfer learning in natural language processing, in the same way that the ImageNet data set is used for pre-training in computer vision.

Method. We follow the protocol of (Conneau et al., 2017) to learn their best model, namely a bi-directional LSTM of about 47M parameters. In particular, the reported results use SGD with an initial learning rate of 0.1 and a hand-designed schedule that adapts to the variations of the validation set: if the validation accuracy does not improve, the learning rate is divided by a factor of 5. We also report results on Adagrad, Adam, AMSGrad and BPGrad. Following the official SGD baseline, Nesterov momentum is deactivated. Using their open-source implementation, we replace the optimization by the DFW algorithm, the CE loss by an SVM, and leave all other components unchanged. In this experiment, we use the conditional gradient direction rather than the CE gradient, since three-way classification does not cause sparsity in the derivative of the hinge loss (which is the issue that originally motivated our use of a different direction). We cross-validate our initial proximal term as a power of ten, and do not manually tune any schedule. In order to disentangle the importance of the loss function from the optimization algorithm, we run the baselines with both an SVM loss and a CE loss. The initial learning rate of the baselines is also cross-validated as a power of ten.

Results. The results are presented in Table 4.3.

Optimizer	Loss	Adagrad	Adam	AMSGrad	BPGrad	DFW	SGD	SGD*
Test Accuracy (%)	CE	83.8	84.5	84.2	83.6	-	84.7	84.5
	SVM	84.6	85.0	85.1	84.2	**85.2**	**85.2**	-

Table 4.3: *Results on the Stanford Natural Language Inference corpus. In black, all adaptive methods have a single hyper-parameter for their step-size. In red, SGD benefits from a hand-designed schedule. SGD* refers to the result reported in (Conneau et al., 2017). The other results have been obtained with their open-source implementation in our own experiments.*

Note that these results outperform the reported testing accuracy of 84.5% in (Conneau et al., 2017) that is obtained with CE. This experiment, which is performed on a completely different architecture and data set than the previous one, confirms that DFW outperforms adaptive gradient methods and matches the performance of SGD with a hand-designed learning rate schedule.

4.6 The Importance of The Step-Size

4.6.1 Impact on Generalization

It is worth discussing the subtle relationship between optimization and generalization. In order to emphasize the impact of implicit regularization, all results presented in this section do not use data augmentation. As a first illustrative example, we consider the following experiment: we take the protocol to train the DN network on CIFAR-100 with SGD, and simply change the initial learning rate to be ten times smaller, and the budget of epochs to be ten times larger. As a result, the final training objective significantly decreases from 0.33 to 0.069. Yet at the same time, the best validation accuracy decreases from 70.94% to 68.7%. A similar effect occurs when decreasing the value of the momentum, and we have observed this across various convolutional architectures. In other words, accurate optimization is less important for generalization than the implicit regularization of a high learning rate.

We have observed DFW to accurately optimize the learning objective in our experiments. However, given the above observation, we believe that its good generalization properties are rather due to its capability to usually maintain a high learning rate at an early stage. Similarly, the good generalization performance of SGD may be due to its schedule with a large number of steps at a high learning rate.

4.6.2 Sensitivity Analysis

The previous section has qualitatively hinted at the importance of the step-size for generalization. Here we quantitatively analyze the impact of the initial learning rate η on both the training accuracy (quality of optimization) and the validation accuracy (quality of generalization). We compare results of the DFW and SGD

algorithms on the CIFAR data sets when varying the value of η as a power of 10. The results on the validation set are summarized in figure 4.4, and the performance on the training set is reported in Appendix C.2.

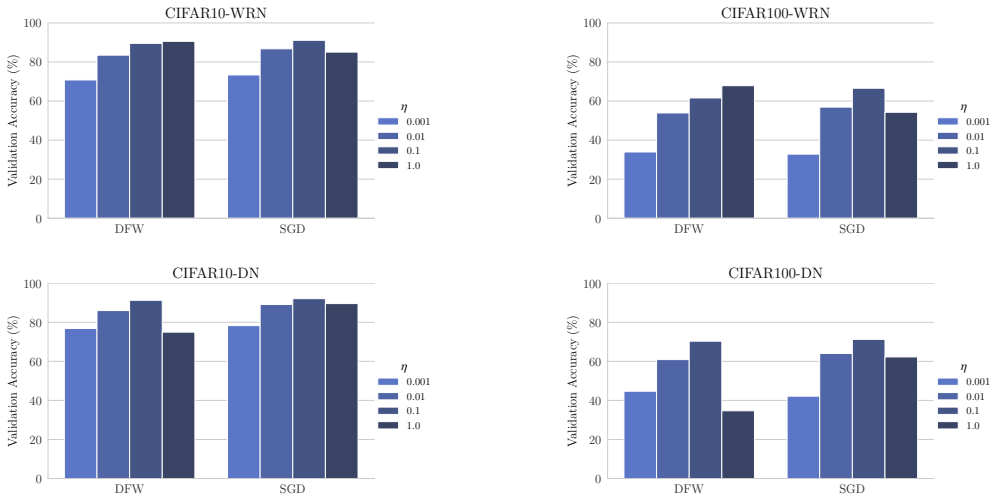


Figure 4.4: *Visualization of the sensitivity analysis for the choice of initial learning rate η on the CIFAR data sets. Each subplot displays the best validation accuracy for DFW and SGD. Similar plots are available in larger format in Appendix C.2.4.*

On the training set, both methods obtain nearly perfect accuracy across at least three orders of magnitude of η (details in Appendix C.2.4). In contrast, the results of figure 4.4 confirm that the validation performance is sensitive to the choice of η for both methods.

In some cases where η is high, SGD obtains a better performance than DFW. This is because the hand-designed schedule of SGD enforces a decay of η, while the DFW algorithm relies on an automatic decay of the step-size γ_t for effective convergence. This automatic decay may not happen if a small proximal term (large η) is combined with a local approximation that is not sufficiently accurate (for instance with a small batch-size).

However, if we allow the DFW algorithm to use a larger batch size, then the local approximation becomes more accurate and it can handle large values of η as well. Interestingly, choosing a larger batch-size and a larger value of η can result in better generalization. For instance, by using a batch-size of 256 (instead of 64)

and $\eta = 1$, DFW obtains a test accuracy of 72.64% on CIFAR-100 with the DN architecture (SGD obtains 70.33% with the settings of (Huang et al., 2017)).

4.6.3 Discussion

Our empirical evidence indicates that the initial learning rate can be a crucial hyper-parameter for good generalization. We have observed in our experiments that such a choice of high learning rate provides a consistent improvement for convolutional neural networks: accurate minimization of the training objective with large initial steps usually leads to good generalization. Furthermore, as mentioned in the previous section, it is sometimes beneficial to even increase the batch-size in order to be able to train the model using large initial steps.

In the case of recurrent neural networks, however, this effect is not as distinct. Additional experiments on different recurrent architectures have showed variations in the impact of the learning rate and in the best-performing optimizer. Further analysis would be required to understand the effects at play.

4.7 Conclusion

We have introduced DFW, an efficient algorithm to train deep neural networks. DFW predominantly outperforms adaptive gradient methods, and obtains similar performance to SGD without requiring a hand-designed learning rate schedule.

We emphasize the generality of our framework in Section 4.3, which enables the training of deep neural networks to benefit from any advance on optimization algorithms for linear SVMs. This framework could also be applied to other loss functions that yield efficiently solvable proximal problems. In particular, our algorithm already supports the use of structured prediction loss functions (Taskar et al., 2003; Tsochantaridis et al., 2004), which can be used, for instance, for image segmentation.

We have mentioned the intricate relationship between optimization and generalization in deep learning. This illustrates a major difficulty in the design of effective optimization algorithms for deep neural networks: the learning objective does not

include all the regularization needed for good generalization. We believe that in order to further advance optimization for deep neural networks, it is essential to alleviate this problem and expose a clear objective function to optimize.

Acknowledgments

This work was supported by the EPSRC grants AIMS CDT EP/L015987/1, Seebibyte EP/M013774/1, EP/P020658/1 and TU/B/000048, and by Yougov. We also thank the Nvidia Corporation for the GPU donation.

5

Training Neural Networks for and by Interpolation

Contents

Abstract

In modern supervised learning, many deep neural networks are able to interpolate the data: the empirical loss can be driven to near zero on all samples simultaneously. In this work, we explicitly exploit this interpolation property for the design of a new optimization algorithm for deep learning. Specifically, we use it to compute an adaptive learning-rate in closed form at each iteration. This results in the Adaptive Learning-rates for Interpolation with Gradients (ALI-G) algorithm. ALI-G retains the main advantage of SGD which is a low computational cost per iteration. But unlike SGD, the learning-rate of ALI-G uses a single constant hyper-parameter and does not require a decay schedule, which makes it considerably easier to tune. We provide convergence guarantees of ALI-G in the stochastic convex setting. Notably, all our convergence results tackle the realistic case where the interpolation property is satisfied up to some tolerance. We provide experiments on a variety of architectures and tasks: (i) learning a differentiable neural computer; (ii) training a wide residual network on the SVHN data set; (iii) training a Bi-LSTM on the SNLI data set; and (iv) training wide residual networks and densely connected networks on the CIFAR data sets. ALI-G produces state-of-the-art results among adaptive methods, and even yields comparable performance with SGD, which requires manually tuned learning-rate schedules. Furthermore, ALI-G is simple to implement in any standard deep learning framework and can be used as a drop-in replacement in existing code. We release PyTorch and Tensorflow implementations of ALI-G as standalone optimizers that can be used as a drop-in replacement in existing code (code available at `https://github.com/oval-group/ali-g`).

5.1 Introduction

Training a deep neural network is a challenging optimization problem: it involves minimizing the average of many high-dimensional non-convex functions. In practice, the main algorithms of choice are Stochastic Gradient Descent (SGD) (Robbins and Monro, 1951) and adaptive gradient methods such as AdaGrad (Duchi et al.,

2011) or Adam (Kingma and Ba, 2015). In recent work, the ability to interpolate – i.e. to achieve near zero loss on all training samples simultaneously – has been used to show convergence of SGD (Ma et al., 2018; Vaswani et al., 2019a; Zhou et al., 2019). This property is usually satisfied in supervised deep learning because of the empirical success of over-parameterized architectures. However, while the convergence analyses provide a better theoretical understanding of SGD, they do not help improve its practical behavior.

In this work, we open a different line of enquiry, namely: *can the interpolation property be used to design a robust and efficient optimization algorithm for deep learning?* In order to answer this question, we begin by giving the following two desiderata of an optimization algorithm for deep learning: (i) an inexpensive computational cost per iteration (typically a call to a stochastic first-order oracle); and (ii) adaptive learning-rates that do not require a manually designed schedule.

We present ALI-G (Adaptive Learning-rates for Interpolation with Gradients), an algorithm that takes advantage of interpolation by design and satisfies both properties mentioned above. Key to the ALI-G algorithm are the following two ideas. First, an adaptive learning-rate can be computed for the non-stochastic gradient direction when the minimum value of the objective function is known (Polyak, 1969; Shor, 1985; Brännlund, 1995; Nedić and Bertsekas, 2001a,b). And second, one such minimum value is usually approximately known for interpolating models: for instance, it is close to zero for a model trained with the cross-entropy loss. By carefully combining these two ideas, we create a stochastic algorithm that provably converges fast in the convex setting and that obtains state-of-the-art results with neural networks.

Procedurally, ALI-G is close to many existing algorithms, such as Deep Frank-Wolfe (Berrada et al., 2019a), aProx (Asi and Duchi, 2019) and L_4 (Rolinek and Martius, 2018). And yet uniquely, thanks to its careful design and analysis, ALI-G enables accurate optimization of a wide class of deep neural networks using only a single hyper-parameter that does not need to be decayed. This makes ALI-G well-suited to the deep learning setting, where hyper-parameter tuning is widely

regarded as an onerous and time consuming task. Since ALI-G is easy to implement in any deep learning framework, we believe that it can prove to be a practical and reliable optimization tool for deep learning.

Contributions. We summarize the contributions of this work as follows:
- We design an optimization algorithm that uses a single hyper-parameter for its learning-rate and does need any decaying schedule. In contrast, the closely related aProx (Asi and Duchi, 2019) and L_4 (Rolinek and Martius, 2018) use respectively two and four hyper-parameters for their learning-rate.
- We provide convergence rates of ALI-G in various stochastic convex settings. Importantly, our theoretical results take into account the error in the estimate of the minimum objective value. To the best of our knowledge, our work is the first to establish convergence rates for interpolation with approximate estimates.
- We demonstrate state-of-the-art results for ALI-G on learning a differentiable neural computer; training variants of residual networks on the SVHN and CIFAR data sets; and training a Bi-LSTM on the Stanford Natural Language Inference data set.

5.2 The Algorithm

5.2.1 Problem Setting

Loss Function. We consider a supervised learning task where the model is parameterized by $\boldsymbol{w} \in \mathbb{R}^p$. Usually, the objective function can be expressed as an expectation over $z \in \mathcal{Z}$, a random variable indexing the samples of the training set:

$$f(\boldsymbol{w}) \triangleq \mathbb{E}_{z \in \mathcal{Z}}[\ell_z(\boldsymbol{w})], \tag{5.1}$$

where each ℓ_z is the loss function associated with the sample z. We assume that each ℓ_z is non-negative, which is the case for the large majority of loss functions used in machine learning. For instance, suppose that the model is a deep neural network with weights $\boldsymbol{w}$ performing classification. Then for each sample z, $\ell_z(\boldsymbol{w})$

can represent the cross-entropy loss, which is always non-negative. Other non-negative loss functions include the structured or multi-class hinge loss, and the $\mathcal{L}_1$ or $\mathcal{L}_2$ loss functions for regression.

Regularization. It is often desirable to employ a regularization function ϕ in order to promote generalization. In this work, we incorporate such regularization as a constraint on the feasible domain: $\Omega = \{\boldsymbol{w} \in \mathbb{R}^p : \ \phi(\boldsymbol{w}) \leq r\}$ for some value of r. In the deep learning setting, this will allow us to assume that the objective function can be driven close to zero without unrealistic assumptions about the regularization. Our framework can handle any constraint set Ω on which Euclidean projections are computationally efficient. This includes the feasible set induced by $\mathcal{L}_2$ regularization: $\Omega = \{\boldsymbol{w} \in \mathbb{R}^p : \ \|\boldsymbol{w}\|_2^2 \leq r\}$, for which the projection is given by a simple rescaling of $\boldsymbol{w}$. Finally, note that if we do not wish to use any regularization, we define $\Omega = \mathbb{R}^p$ and the corresponding projection is the identity.

Problem Formulation. The learning task can be expressed as the problem $(\mathcal{P})$ of finding a feasible vector of parameters $\mathbf{w}_\star \in \Omega$ that minimizes f:

$$\mathbf{w}_\star \in \operatorname*{argmin}_{\boldsymbol{w} \in \Omega} f(\boldsymbol{w}). \tag{$\mathcal{P}$}$$

Also note that $f_\star$ refers to the minimum value of f over Ω: $f_\star \triangleq \min_{\boldsymbol{w} \in \Omega} f(\boldsymbol{w})$.

5.2.2 The Polyak Step-Size

Before outlining the ALI-G algorithm, we begin with a brief description of the Polyak step-size, from which ALI-G draws some fundamental ideas.

Setting. We assume that $f_\star$ is known and we use non-stochastic updates: at each iteration, the full objective f and its derivative are evaluated. We denote by $\nabla f(\boldsymbol{w})$ the first-order derivative of f at $\boldsymbol{w}$ (e.g. $\nabla f(\boldsymbol{w})$ can be a sub-gradient or the gradient). In addition, $\|\cdot\|$ is the standard Euclidean norm in $\mathbb{R}^p$, and $\Pi_\Omega(\boldsymbol{w})$ is the Euclidean projection of the vector $\boldsymbol{w} \in \mathbb{R}^p$ on the set Ω.

Polyak Step-Size. At time-step t, using the Polyak step-size (Polyak, 1969; Shor, 1985; Bränlund, 1995; Nedić and Bertsekas, 2001a,b) yields the following update:

$$\boldsymbol{w}_{t+1} = \Pi_{\Omega}\left(\boldsymbol{w}_t - \gamma_t \nabla f(\boldsymbol{w}_t)\right), \text{ where } \gamma_t \triangleq \frac{f(\boldsymbol{w}_t) - f_\star}{\|\nabla f(\boldsymbol{w}_t)\|^2}, \tag{5.2}$$

where we loosely define $\frac{0}{0} = 0$ for simplicity purposes.

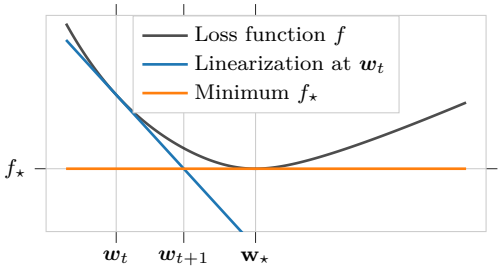


Figure 5.1: *Illustration of the Polyak step-size in 1D. In this case, and further assuming that $f_\star = 0$, the algorithm coincides with the Newton-Raphson method for finding roots of a function.*

Interpretation. It can be shown that $\boldsymbol{w}_{t+1}$ lies on the intersection between the linearization of f at $\boldsymbol{w}_t$ and the horizontal plane $z = f_\star$ (see Figure 5.1, more details in Proposition 18 in the supplementary material). Note that since $f_\star$ is the minimum of f, the Polyak step-size γ_t is necessarily non-negative.

Limitations. Equation (5.2) has two major short-comings that prevent its applicability in a machine learning setting. First, each update requires a full evaluation of f and its derivative. Stochastic extensions have been proposed in (Nedić and Bertsekas, 2001a,b), but they still require frequent evaluations of f. This is expensive in the large data setting, and even computationally infeasible when using massive data augmentation. Second, when applying this method to the non-convex setting of deep neural networks, the method sometimes fails to converge.

Therefore we would like to design an extension of the Polyak step-size that (i) is inexpensive to compute in a stochastic setting (e.g. with a computational cost that is independent of the total number of training samples), and (ii) converges in practice when used with deep neural networks. The next section introduces the ALI-G algorithm, which achieves these two goals in the interpolation setting.

5.2.3 The ALI-G Algorithm

We now present the ALI-G algorithm. For this, we suppose that we are in an interpolation setting: the model is assumed to be able to drive the loss function to near zero on all samples simultaneously.

Algorithm. The main steps of the ALI-G algorithm are provided in Algorithm 5. ALI-G iterates over three operations until convergence. First, it computes a stochastic approximation of the learning objective and its derivative (line 3). Second, it computes a step-size decay parameter γ_t based on the stochastic information (line 4). Third, it updates the parameters by moving in the negative derivative direction by an amount specified by the step-size and projecting the resulting vector on to the feasible region (line 5).

Algorithm 5 *The ALI-G algorithm*

Require: maximal learning-rate η, initial feasible $\boldsymbol{w}_0 \in \Omega$, small constant $\delta > 0$

1: $t = 0$
2: **while** not converged **do**
3: Get $\ell_{z_t}(\boldsymbol{w}_t)$, $\nabla \ell_{z_t}(\boldsymbol{w}_t)$ with z_t drawn i.i.d.
4: $\gamma_t = \min\left\{\frac{\ell_{z_t}(\boldsymbol{w}_t)}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 + \delta}, \eta\right\}$
5: $\boldsymbol{w}_{t+1} = \Pi_\Omega\left(\boldsymbol{w}_t - \gamma_t \nabla \ell_{z_t}(\boldsymbol{w}_t)\right)$
6: $t = t + 1$
7: **end while**

Comparison with the Polyak Step-Size. There are three main differences to the update in equation (5.2). First, each update only uses the loss ℓ_{z_t} and its derivative rather than the full objective f and its derivative. Second, the learning-rate γ_t is clipped to η, the maximal learning-rate hyper-parameter. We emphasize that η remains constant throughout the iterations, therefore it is a single hyper-parameter and does not need a schedule like SGD learning-rate. Third, the minimum $f_\star$ has been replaced by the lower-bound of 0. All these modifications will be justified in the next section.

The ALI-G$^\infty$ Variant. When ALI-G uses no maximal learning-rate, we refer to the algorithm as ALI-G$^\infty$, since it is equivalent to use an infinite maximal learning-rate. Note that ALI-G$^\infty$ requires no hyper-parameter for its step-size.

5.3 Justification and Analysis

5.3.1 Interpolation Allows Inexpensive Stochastic Updates

By definition, the interpolation setting gives $f_\star = 0$, which we used in ALI-G to simplify the formula of the learning-rate γ_t. More subtly, the interpolation property 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, each training sample can use its own learning-rate without harming progress on the other ones. Recall that ALI-G$^\infty$ is the variant of ALI-G that uses no maximal learning-rate. The following result formalizes the convergence guarantee of ALI-G$^\infty$ in the stochastic convex setting:

Theorem 1 (Convex and Lipschitz)**.** *We assume that Ω is a convex set, and that for every $z \in \mathcal{Z}$, ℓ_z is convex and C-Lipschitz. Let $\mathbf{w}_\star$ be a solution of ($\mathcal{P}$), and assume that the interpolation property is approximately satisfied: $\forall z \in \mathcal{Z}$, $\ell_z(\mathbf{w}_\star) \leq \varepsilon$, for some interpolation tolerance $\varepsilon \geq 0$. Then ALI-G$^\infty$ applied to f satisfies:*

$$f\left(\frac{1}{T+1}\sum_{t=0}^{T}\boldsymbol{w}_t\right) \leq \varepsilon\sqrt{\left(\frac{C^2}{\delta}+1\right)} + \frac{\|\boldsymbol{w}_0 - \mathbf{w}_\star\|\sqrt{C^2+\delta}}{\sqrt{T+1}}. \tag{5.3}$$

In other words, by assuming interpolation, ALI-G provably converges while requiring only $\ell_{z_t}(\boldsymbol{w}_t)$ and $\nabla\ell_{z_t}(\boldsymbol{w}_t)$ (stochastic estimation per sample) to compute its learning-rate. In contrast, the Polyak step-size, which does not exploit interpolation, would require $f(\boldsymbol{w}_t)$ and $\nabla f(\boldsymbol{w}_t)$ to compute the learning-rate (deterministic computation over all training samples). This constitutes a major computational advantage of ALI-G over the usual Polyak step-size.

We emphasize that in Theorem 1, our careful analysis explicitly shows the dependency of the convergence result on the interpolation tolerance ε. It is

reassuring to note that convergence is exact when the interpolation property is exactly satisfied ($\varepsilon = 0$).

In the supplementary material, we also establish convergence rates of $\mathcal{O}(1/T)$ for smooth convex functions, and $\mathcal{O}(\exp(-\alpha T/8\beta))$ for α-strongly convex and β-smooth functions. Similar results can be proved when using a maximal learning-rate η: the convergence speed then remains unchanged provided that η is large enough, and it is lowered when η is small. We refer the interested reader to the supplementary for the formal results and their proofs.

5.3.2 A Maximal Learning-Rate Helps with Non-Convexity

The Polyak step-size may fail to converge when the objective is non-convex, as figure 5.2 illustrates: in this (non-convex) setting, gradient descent with Polyak step-size oscillates between two symmetrical points because its step-size is too large (details in the supplementary).

Indeed, we empirically find that non-convexity usually leads to overestimation of the Polyak step-size. Intuitively, using a maximal learning-rate allows ALI-G to behave like constant step-size SGD in non-convex regions where the Polyak step-size would be over-estimated, and to automatically use the Polyak step-size once reaching a convex basin of convergence.

Importantly, using a maximal learning-rate can be seen as a very natural extension of SGD when using a non-negative loss function:

Proposition 8. *[Proximal Interpretation] Suppose that $\Omega = \mathbb{R}^p$ and let $\delta = 0$. We consider the update performed by SGD: $\boldsymbol{w}_{t+1}^{SGD} = \boldsymbol{w}_t - \eta_t \nabla \ell_{z_t}(\boldsymbol{w}_t)$; and the update performed by ALI-G: $\boldsymbol{w}_{t+1}^{ALI\text{-}G} = \boldsymbol{w}_t - \gamma_t \nabla \ell_{z_t}(\boldsymbol{w}_t)$, where $\gamma_t = \min\left\{\frac{\ell_{z_t}(\boldsymbol{w}_t)}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 + \delta}, \eta\right\}$. Then we have:*

$$\boldsymbol{w}_{t+1}^{SGD} = \underset{\boldsymbol{w}\in\mathbb{R}^p}{\operatorname{argmin}} \left\{ \frac{1}{2\eta_t}\|\boldsymbol{w} - \boldsymbol{w}_t\|^2 + \ell_{z_t}(\boldsymbol{w}_t) + \nabla \ell_{z_t}(\boldsymbol{w}_t)^\top (\boldsymbol{w} - \boldsymbol{w}_t) \right\}, \tag{5.4}$$

$$\boldsymbol{w}_{t+1}^{ALI\text{-}G} = \underset{\boldsymbol{w}\in\mathbb{R}^p}{\operatorname{argmin}} \left\{ \frac{1}{2\eta}\|\boldsymbol{w} - \boldsymbol{w}_t\|^2 + \max\left\{ \ell_{z_t}(\boldsymbol{w}_t) + \nabla \ell_{z_t}(\boldsymbol{w}_t)^\top (\boldsymbol{w} - \boldsymbol{w}_t), 0 \right\} \right\}. \tag{5.5}$$

In other words, at each iteration, ALI-G solves a proximal problem in closed form in a similar way to SGD. In both cases, the loss function ℓ_{z_t} is locally approximated by a first-order Taylor expansion at $\boldsymbol{w}_t$. The difference is that ALI-G also exploits the fact that ℓ_{z_t} is non-negative. This allows ALI-G to use a constant value for η in the interpolation setting, while the learning-rate η_t of SGD needs to be manually decayed.

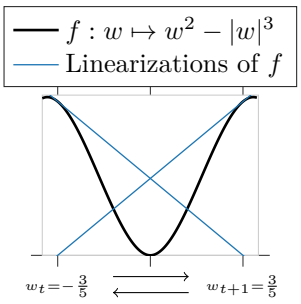


Figure 5.2: *A simple example where the Polyak step-size oscillates due to non-convexity. On this problem, ALI-G converges whenever its maximal learning-rate η is lower than* 10.

It is currently an open question whether ALI-G can be proved to converge in the stochastic non-convex setting given a sufficiently small yet constant maximal learning-rate η.

5.4 Related Work

Interpolation in Deep Learning. As mentioned in the introduction, recent works have successfully exploited the interpolation assumption to prove convergence of SGD in the context of deep learning (Ma et al., 2018; Vaswani et al., 2019a; Zhou et al., 2019). Such works are complementary to ours in the sense that they provide a convergence analysis of an existing algorithm for deep learning.

Adaptive Gradient Methods. Similarly to ALI-G, most adaptive gradient methods also rely on tuning a single hyper-parameter, thereby providing a more pragmatic alternative to SGD that needs a specification of the full learning-rate schedule. While the most popular ones are Adagrad (Duchi et al., 2011), RMSPROP (Tieleman and Hinton, 2012), Adam (Kingma and Ba, 2015) and AMSGrad (Reddi et al., 2018), there have been many other variants (Zeiler, 2012; 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). However, as pointed out in (Wilson et al., 2017), adaptive gradient methods tend to give poor generalization in supervised learning. In our experiments, the results provided by ALI-G are significantly better than those obtained by the most popular adaptive gradient methods. Recently, Liu et al. (2019) have proposed to "rectify" Adam with a learning-rate warmup, which partly bridges the gap in generalization performance between Adam and SGD. However, their method still requires a learning-rate schedule, and thus remains difficult to tune on new tasks.

Adaptive Learning-Rate Algorithms. Vaswani et al. (2019b) show that one can use line search in a stochastic setting for interpolating models while guaranteeing convergence. However, in contrast to our work, the resulting algorithm requires more than one hyper-parameter (up to four), and the line-search is not computed in closed form. Less closely related methods, included second-order ones, adaptively compute the learning-rate without using the minimum (Schaul et al., 2013; Martens and Grosse, 2015; Tan et al., 2016; Zhang et al., 2017c; Baydin et al., 2018; Wu et al., 2018; Li and Orabona, 2019; Henriques et al., 2019), but do not demonstrate competitive generalization performance against SGD with a well-tuned hand-designed schedule.

L_4 Algorithm. The L_4 algorithm (Rolinek and Martius, 2018) also uses a modified version of the Polyak step-size. However, the L_4 algorithm computes an online estimate of $f_\star$ rather than relying on a fixed value. This requires three hyper-parameters, which are in practice sensitive to noise and crucial for empirical convergence of the method. In addition, L_4 does not come with convergence guarantees. In contrast, by utilizing the interpolation property and a maximal learning-rate, our method is able to (i) provide reliable and accurate minimization with only a single hyper-parameter, and (ii) offer guarantees of convergence in the stochastic convex setting.

Frank-Wolfe Methods. The proximal interpretation in Proposition 8 allows us to draw additional parallels to existing methods. In particular, the formula of the learning-rate γ_t may remind the reader of the Frank-Wolfe algorithm (Frank

and Wolfe, 1956) in some of its variants (Locatello et al., 2017), or other dual methods (Lacoste-Julien and Jaggi, 2013; Shalev-Shwartz and Zhang, 2016). This is because such methods solve in closed form the dual of problem (5.5), and problems in the form of (5.5) naturally appear in dual coordinate ascent methods (Shalev-Shwartz and Zhang, 2016).

When no regularization is used, ALI-G and Deep Frank-Wolfe (DFW) (Berrada et al., 2019a) are procedurally identical algorithms. This is because in such a setting, one iteration of DFW also amounts to solving (5.5) in closed-form – more generally, DFW is designed to train deep neural networks by solving proximal linear support vector machine problems approximately. However, we point out the two fundamental advantages of ALI-G over DFW: (i) ALI-G can handle arbitrary (lower-bounded) loss functions, while DFW can only use convex piece-wise linear loss functions; and (ii) as seen previously, ALI-G provides convergence guarantees in the convex setting.

SGD with Polyak's Learning-Rate. (Oberman and Prazeres, 2019) extend the Polyak step-size to rely on a stochastic estimation of the gradient $\nabla \ell_{z_t}(\boldsymbol{w}_t)$ only, instead of the expensive deterministic gradient $\nabla f(\boldsymbol{w}_t)$. However, they still require to evaluate $f(\boldsymbol{w}_t)$, the objective function over the entire training data set, in order to compute its learning-rate, which makes the method impractical. In addition, since they do not do exploit the interpolation setting nor the fact that regularization can be expressed as a constraint, they also require the optimal objective function value $f_\star$ in advance.

aProx **Algorithm.** (Asi and Duchi, 2019) have recently introduced the aProx algorithm, a family of proximal stochastic optimization algorithms for convex problems. Notably, the aProx "truncated model" version is similar to ALI-G. However, there are four clear advantages of our work over (Asi and Duchi, 2019) in the interpolation setting, in particular for training neural networks. First, our work is the first to empirically demonstrate the applicability and usefulness of the algorithm on varied modern deep learning tasks – most of our experiments use several orders of magnitude more data and model parameters than the small-scale convex problems

of (Asi and Duchi, 2019). Second, our analysis and insights allow us to make more aggressive choices of learning rate than (Asi and Duchi, 2019). Indeed, (Asi and Duchi, 2019) assume that the maximal learning-rate is exponentially decaying, even in the interpolating convex setting. In contrast, by avoiding the need for an exponential decay, the learning-rate of ALI-G requires only one hyper-parameters instead of two for aProx. Third, our analysis takes into account the interpolation tolerance $\varepsilon \geq 0$ rather than unrealistically assuming the perfect case $\varepsilon = 0$ (that would require infinite weights when using the cross-entropy loss for instance). Fourth, our analysis proves fast convergence in function space rather than iterate space.

5.5 Experiments

We empirically compare ALI-G to the optimization algorithms most commonly used in deep learning. Our experiments span a variety of architectures and tasks: (i) learning a differentiable neural computer; (ii) training wide residual networks on SVHN; (iii) training a Bi-LSTM on the Stanford Natural Language Inference data set; and (iv) training wide residual networks and densely connected networks on the CIFAR data sets. 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 at standardizing baselines for deep learning optimizers. 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. Despite this, ALI-G obtains competitive performance with SGD. In addition, ALI-G is the best performing method with a single hyper-parameter on the difficult tasks of Bi-LSTM on SNLI and ResNet variants on CIFAR.

The code to reproduce our results will be made publicly available. In the Tensorflow (Abadi et al., 2015) experiment, we use the official and publicly available implementation of L_4[1]. In the PyTorch (Paszke et al., 2017) experiments, we use our implementation of L_4, which we unit-test against the official Tensorflow

[1] `https://github.com/martius-lab/l4-optimizer`

implementation. In addition, we employ the official implementation of DFW[2] and we re-use their code for the experiments on SNLI and CIFAR. All experiments are performed either on a 12-core CPU (differentiable neural computer) or on a single GPU (SVHN, SNLI, CIFAR).

5.5.1 Differentiable Neural Computers

Setting. The Differentiable Neural Computer (DNC) (Graves et al., 2016) is a recurrent neural network that aims at performing computing tasks by learning from examples rather than by executing an explicit program. In this case, the DNC learns to repeatedly copy a fixed size string given as input. Although this learning task is relatively simple, the complex architecture of the DNC makes it an interesting benchmark problem for optimization algorithms.

Methods. We use the official and publicly available implementation of DNC[3]. We vary the initial learning rate as powers of ten between 10^{-4} and 10^{4} for each method except for L_4Adam and L_4Mom. For L_4Adam and L_4Mom, since the main hyper-parameter α is designed to lie in $(0, 1)$, we vary it between 0.05 and 0.095 with a step of 0.1. The gradient norm is clipped for all methods except for ALI-G, L_4Adam and L_4Mom (as recommended by (Rolinek and Martius, 2018)).

Results. We present the results in Figure 5.3. ALI-G provides accurate optimization for any η within $[10^{-1}, 10^{6}]$, and is among the best performing methods by reaching an objective function of 4.10^{-8}. On this task, RMSProp, L_4Adam and L_4Mom also provide accurate and robust optimization. In contrast to ALI-G and the L_4 methods, the most commonly used algorithms such as SGD, SGD with momentum and Adam are very sensitive to their main learning-rate hyper-parameter. Note that the difference between well-performing methods is not significant here because these reach the numerical precision limit of single-precision float numbers.

[2] `https://github.com/oval-group/dfw`
[3] `https://github.com/deepmind/dnc`

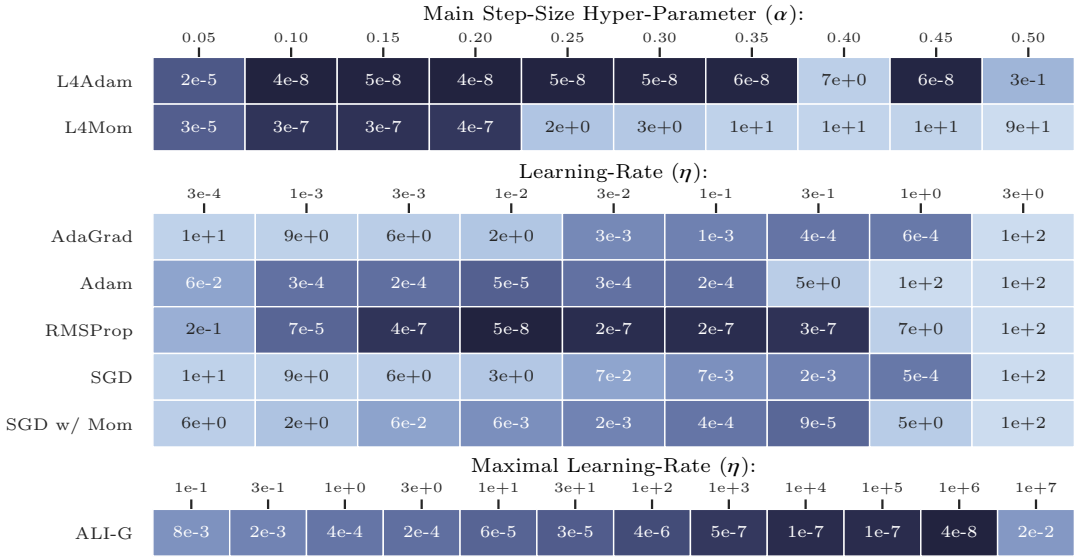


Figure 5.3: *Final objective function when training a Differentiable Neural Computer for* $10k$ *steps (lower is better). The intensity of each cell is log-proportional to the value of the objective function (darker is better). ALI-G obtains good performance for a very large range of* η *(*$10^{-1} \leq \eta \leq 10^{6}$*).*

5.5.2 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 hyper-parameter 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 hyper-parameter is tuned as a power of 10. The $\mathcal{L}_2$ regularization is cross-validated in $\{0.0001, 0.0005\}$ for all methods but ALI-G. For ALI-G, the regularization is expressed as a constraint on the $\mathcal{L}_2$-norm of the parameters, and its maximal value is set to 50. SGD, ALI-G 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).

Results. The results are presented in Table 5.1. On this relatively easy task, most methods achieve about 98% test accuracy. Despite the cross-validation, L_4Mom does not converge on this task. Even though SGD benefits from a hand-designed schedule, ALI-G and other adaptive methods obtain close performance to it.

Adagrad	Adam	AMSGrad	BPGrad	DFW	L_4Adam	L_4Mom	ALI-G	SGD	SGD†
98.0	97.9	97.9	98.1	98.1	**98.2**	19.6	98.1	98.3	98.4

Table 5.1: *Test Accuracy (%) on SVHN. In red, SGD benefits from a hand-designed schedule for its learning-rate. In black, adaptive methods, including ALI-G, have a single hyper-parameter for their learning-rate. SGD† refers to the performance reported by (Zagoruyko and Komodakis, 2016).*

5.5.3 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 in 570k pairs of sentences, with each pair labeled as entailment, neutral or contradiction. This large scale data set is commonly used as a pre-training corpus for transfer learning to many other natural language tasks where labeled data is scarcer (Conneau et al., 2017) – much like ImageNet is used for pre-training in computer vision. We follow the protocol of our earlier work in (Berrada et al., 2019a); we also re-use our results for the baselines.

Method. For L_4Adam and L_4Mom, the main hyper-parameter 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 schedule. The value of the main hyper-parameter

η is cross-validated as a power of ten for the ALI-G algorithm and for previously reported adaptive methods. Following the implementation by (Conneau et al., 2017), no $\mathcal{L}_2$ regularization 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 SVMs only. For all optimization algorithms, the model is trained for 20 epochs, following (Conneau et al., 2017).

Loss	Adagrad*	Adam*	AMSGrad*	BPGrad*	DFW*	L4Adam	L4Mom	ALI-G$^\infty$	ALI-G	SGD*	SGD†
CE	83.8	84.5	84.2	83.6	-	83.3	83.7	84.6	**84.8**	84.7	84.5
SVM	84.6	85.0	85.1	84.2	**85.2**	82.5	83.2	84.7	**85.2**	85.2	-

Table 5.2: *Test Accuracy (%) on SNLI. In red, SGD benefits from a hand-designed schedule for its learning-rate. In black, adaptive methods have a single hyper-parameter for their learning-rate. In blue, ALI-G$^\infty$ does not have any hyper-parameter for its 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* re-use results from (Berrada et al., 2019a). SGD† is the result from (Conneau et al., 2017).*

Results. We present the results in Table 5.2. ALI-G$^\infty$ is the only method that requires no hyper-parameter for its learning-rate. Despite this, and the fact that SGD employs a learning-rate schedule that has been hand designed for good validation performance, ALI-G$^\infty$ is still able to obtain results that are competitive with SGD. Moreover, ALI-G, which requires a single hyper-parameter for the learning-rate, outperforms all other methods for both the SVM and the CE loss functions.

5.5.4 Wide Residual Networks and Densely Connected Networks on CIFAR

Setting. We follow the methodology of our earlier work (Berrada et al., 2019a), and we reproduce our previous results. 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 centered and normalized per channel. We apply standard data augmentation with random horizontal flipping and random crops. AMSGrad

was selected in (Berrada et al., 2019a) because it was the best adaptive method on similar tasks, outperforming in particular Adam and Adagrad. In addition to the baselines from (Berrada et al., 2019a), we also provide the performance of L_4Adam, L_4Mom, AdamW (Loshchilov and Hutter, 2019) and Yogi (Zaheer et al., 2018).

Method. All optimization methods employ the cross-entropy loss, except for the DFW algorithm, which is designed to use an SVM loss. For DN and WRN respectively, SGD uses the manual learning rate schedules from (Huang et al., 2017) and (Zagoruyko and Komodakis, 2016). Following our previous work (Berrada et al., 2019a), 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 hyper-parameter α is cross-validated in $\{0.015, 0.15\}$. For AMSGrad, AdamW, Yogi, DFW and ALI-G, the learning-rate hyper-parameter η is cross-validated as a power of 10 (in practice $\eta \in \{0.1, 1\}$ for ALI-G). SGD, DFW and ALI-G use a Nesterov momentum of 0.9. Following our previous work (Berrada et al., 2019a), for all methods but ALI-G and AdamW, the $\mathcal{L}_2$ regularization 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, $\mathcal{L}_2$ regularization is expressed as a constraint on the norm on the vector of parameters; its maximal value is set to 100 for the WRN models, 80 for DN on CIFAR-10 and 75 for DN on CIFAR-100. For all optimization algorithms, the WRN model is trained for 200 epochs and the DN model for 300 epochs, following respectively (Zagoruyko and Komodakis, 2016) and (Huang et al., 2017).

Results. We present the results in Table 5.3. In this setting again, ALI-G obtains competitive performance with manually decayed SGD. ALI-G largely outperforms AMSGrad, AdamW and Yogi. In addition, it significantly bridges the gap between DFW and SGD on CIFAR-10 with the WRN model, and on CIFAR-100 with the DN one.

Data Set	Architecture	AMSGrad	AdamW	Yogi	DFW	L_4Adam	L_4Mom	ALI-G	SGD	SGD†
CIFAR-10	WRN	90.8	92.1	91.2	94.2	90.5	91.6	**95.2**	95.3	95.4
	DN	91.7	92.6	92.1	94.6	90.8	91.9	**95.0**	95.1	-
CIFAR-100	WRN	68.7	69.6	68.7	**76.0**	61.7	61.4	75.8	77.8	78.8
	DN	69.4	69.5	69.6	73.2	60.5	62.6	**76.3**	76.3	-

Table 5.3: *Test Accuracy (%) on the CIFAR data sets. In red, SGD benefits from a hand-designed schedule for its learning-rate. In black, adaptive methods, including ALI-G, have a single hyper-parameter for their learning-rate. SGD† refers to the result from (Zagoruyko and Komodakis, 2016). Each reported result is an average over three independent runs; the standard deviations are reported in Appendix (they are at most* 0.3 *for ALI-G and SGD).*

5.5.5 Training Performance

The experiments have so far focused on testing accuracy (except for the DNC task), because that is the main metric driving practitioners' choice of optimization algorithm. In this section, we empirically assess the performance of ALI-G and its competitors in terms of training objective. In order to have comparable objective functions, the $\mathcal{L}_2$ regularization is deactivated. We do not use dropout. 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. All methods use a fix budget of 160 epochs for WRN-SVHN, 200 epochs for WRN-CIFAR and 300 epochs for DN-CIFAR, following (Zagoruyko and Komodakis, 2016) and (Huang et al., 2017). The L_4 methods diverge on CIFAR-100 in this setting. For clarity, we only display the performance of SGD, Adam, Adagrad and ALI-G (DFW does not support the cross-entropy loss). Here SGD uses a constant learning-rate to emphasize the need for adaptivity. Therefore all methods use one hyper-parameter for their learning-rate. As can be seen, ALI-G provides better training performance than the baseline algorithms on all tasks.

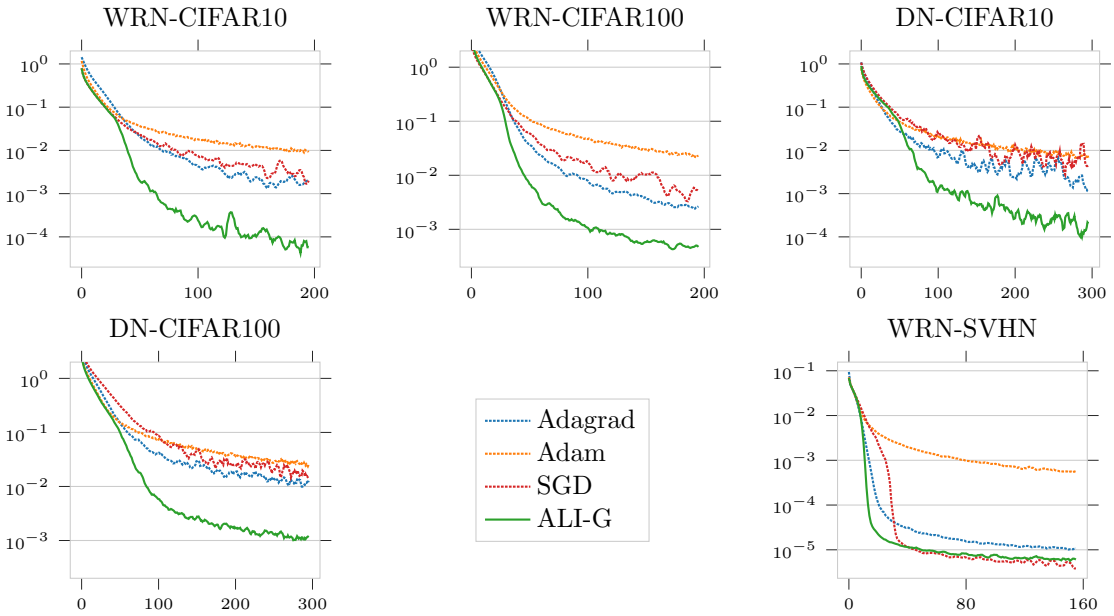


Figure 5.4: *Objective function over the epochs on CIFAR-10, CIFAR-100 and SVHN (smoothed with a moving average over 5 epochs). On SVHN, ALI-G obtains similar performance to its competitors and converges faster. On CIFAR-10 and CIFAR-100, which are more difficult tasks, ALI-G yields an objective function that is an order of magnitude better than the baselines.*

5.6 Discussion

We hope that the ALI-G algorithm is a helpful step towards efficient and reliable training of deep neural networks. ALI-G is readily applicable to a broad range of applications in deep learning where the model can interpolate the data. When that is not the case however, it would be interesting to design new algorithms that adapt the minimum $f_\star$ online while requiring few hyper-parameters. This could be achieved for instance by building upon the works of (Nedić and Bertsekas, 2001b) and (Rolinek and Martius, 2018).

6
Summary and Extensions

Contents

6.1 Summary

In this thesis, we tackle two aspects of optimization for deep learning: how to formulate a loss function, and how to minimize a learning objective. Before discussing possible extensions in future research, we summarize the results of each chapter below.

In chapter 2, we show how to efficiently smooth a top-k hinge loss function by exploiting a connection to polynomial algebra. On top-k classification tasks, this results in a loss function that is less prone to over-fitting than cross-entropy, in particular when there are a few samples or when there is label confusion.

In chapter 3, we design a layer-wise optimization algorithm for piecewise linear CNNs. We exploit the difference-of-convex structure of the problem to create local (convex) linear SVM problems, which we solve with the Block-Coordinate Frank-Wolfe algorithms. This allows us to guarantee monotonic improvement of the objective function, and in practice to improve over state-of-the-art adaptive gradient methods.

In chapter 4, we introduce the Deep Frank-Wolfe algorithm, which leverages compositionality of the neural network and the loss to create proximal problems. Each proximal problem is a linear SVM, which we solve approximately with the Frank-Wolfe algorithm. Empirically, we demonstrate that DFW outperforms adaptive gradient methods, and provides comparable performance to SGD, while avoiding the need for a learning-rate schedule.

In chapter 5, we detail an optimization algorithm termed Adaptive Learning-rate for Interpolation with Gradients (ALI-G). ALI-G exploits the interpolation property to derive a learning-rate in closed-form at each iteration. We prove convergence of ALI-G in various stochastic convex settings, and we demonstrate state-of-the-art performance for the training of interpolating models, while requiring a single hyper-parameter for the learning-rate.

6.2 Discussion

We now turn to a critical discussion of the different methods presented in this thesis, in hindsight and in light of work that has appeared since their publication.

Smooth Loss Functions for Deep Top-k Classsification. Since publication of (Berrada et al., 2018) (corresponding to chapter 2), Amos et al. (2019) have made two interesting points that complement our investigation.

First, they empirically showed that the summation algorithm can be faster than our divide-and-conquer approach, in particular when k is large. This can be further understood by analyzing the theoretical time complexity of the two algorithms when taking into account parallelism. Indeed, let us denote by n the number of classes, and let us assume a hardware model with infinite parallelization capability. Then, as we detail in chapter 2, the divide-and-conquer approach can be computed in $\mathcal{O}(k \log(n))$ time (the sequential time complexity $\mathcal{O}(kn)$ is reduced by a factor of $\log(n)$ thanks to parallelization). In contrast, it can be shown that the summation algorithm can be computed in $\mathcal{O}(n)$ time (the sequential time complexity $\mathcal{O}(kn)$ is reduced by a factor of k thanks to parallelization). As a result, the summation algorithm and the divide-an-conquer approaches will be optimal in different regimes of (k, n): the summation algorithm is faster for large values of k and small values of n, while the divide-and-conquer approach is faster for small k and large n.

Second, they show that other top-k loss functions provide competitive performance on noisy CIFAR-100 and subsets of ImageNet at a lower computational cost. This establishes faster baselines for top-k classification. Interestingly, this also confirms that the source of improvements we observed in chapter 2 is due to the use of a surrogate for the top-k prediction loss (rather than any other property of the smoothed loss function).

Nevertheless, it is worth noting that the smoothing proposed in chapter 2 can be applied to any (continuous) piecewise linear loss function. Consequently, our insights about smoothing may be useful for many learning tasks other than top-k classification. However, the resulting computational challenges to evaluate the

smoothed loss function may be specific to each use case. It would be interesting to characterize "easy" problems, for which the approach can be reduced to polynomial expansion as we have done in the top-k case.

Trusting SVMs for Piecewise-Linear CNNs. The method presented in chapter 3 trains only a layer at a time. This may result in slow training for deep neural networks with many layers. The Deep Frank-Wolfe algorithm (chapter 4), which creates a linear SVM including all layers simultaneously, can be seen as a solution that overcomes this problem.

The main advantage of chapter 3 is that it guarantees convergence to a stationary point with monotonic improvements of the learning objective, while requiring little tuning of the hyper-parameters. This makes the algorithm robust to widely different applications, and it may be worth investigating its usefulness for the fine-tuning of pre-trained models for transfer learning. Indeed, the layer-wise approach may be particularly appropriate for this setting for two reasons. First, training begins by exactly solving the convex problem of training the last layer, which should permit very fast improvement at an early stage. Second, the method automatically uses different learning-rates per layer, which may be useful in transfer learning. Indeed, in such a case, layers closer to the input may not need to change as much as layers close to the output. It would be interesting to test these hypothetical insights in practice.

Deep Frank-Wolfe For Neural Network Optimization. The Deep Frank-Wolfe (DFW) algorithm performs only one step per proximal problem for efficiency reasons. It would be interesting to investigate how performing more steps per proximal problem impacts performance in practice. In order to do so, one needs to access to the gradient of a linearized model at a different point than the linearization point. This corresponds exactly to calculating a Jacobian-vector product. It can be done as the same cost as a forward pass if we have access to forward-mode automatic differentiation (Nocedal and Wright, 2006). With access to backward-mode automatic differentiation only, this can be performed with a double backward pass (Townsend, 2017).

As mentioned at the end of chapter 4, DFW can make use of any SVM solver to solve the proximal problem. This means that DFW could also benefit from further advances in optimization algorithms for SVMs.

Training Neural Networks for and by Interpolation. The ALI-G algorithm exploits the interpolation property to derive an inexpensive adaptive learning-rate in closed-form. A natural extension would be to estimate the minimum online rather than to rely on the interpolation property. This could be done by building upon the existing approaches of (Nedić and Bertsekas, 2001b) and (Rolinek and Martius, 2018).

It would also be interesting to explore the applicability of ALI-G to other descent directions, such as the ones provided by adaptive gradient methods.

Finally, it would be interesting to probe the performance of adaptive gradient methods when the preconditioning vector is summed to a scalar value, so as to preserve the stochastic gradient as the descent direction. Specifically, it would be interesting to see if this results in efficient adaptation of the learning-rate while preserving the good generalization performance of SGD.

Appendices

A

Smooth Loss Functions for Deep Top-k Classification

Contents

A.1 Surrogate Losses: Properties

In this section, we fix n the number of classes. We let $\tau > 0$ and $k \in \{1, ..., n-1\}$. All following results are derived with a loss l_k defined as in equation (2.8):

$$l_k(\mathbf{s}, y) \triangleq \max\left\{\left(\frac{1}{k}\mathbf{s}_{\setminus y} + \mathbf{1}\right)_{[k]} - \frac{1}{k}s_y, 0\right\}. \tag{A.1}$$

A.1.1 Reformulation

Proposition 9. *We can equivalently re-write l_k as:*

$$l_k(\mathbf{s}, y) = \max_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}} \left\{\Delta_k(\bar{\mathbf{y}}, y) + \frac{1}{k}\sum_{j \in \bar{\mathbf{y}}} s_j\right\} - \max_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \left\{\frac{1}{k}\sum_{j \in \bar{\mathbf{y}}} s_j\right\}. \tag{A.2}$$

Proof.

$$\begin{aligned}
l_k(\mathbf{s}, y) &= \max\left\{\left(\frac{1}{k}\mathbf{s}_{\setminus y} + \mathbf{1}\right)_{[k]} - \frac{1}{k}s_y, 0\right\}, \\
&= \max\left\{\left(\frac{1}{k}\mathbf{s}_{\setminus y} + \mathbf{1}\right)_{[k]} - \frac{1}{k}s_y, 0\right\} + \left(\frac{1}{k}\sum_{j=1}^{k-1} s_{[j]} + \frac{1}{k}s_y\right) \\
&\quad - \left(\frac{1}{k}\sum_{j=1}^{k-1} s_{[j]} + \frac{1}{k}s_y\right), \\
&= \max\left\{\left(\frac{1}{k}\mathbf{s}_{\setminus y} + \mathbf{1}\right)_{[k]} + \frac{1}{k}\sum_{j=1}^{k-1} s_{[j]}, \frac{1}{k}\sum_{j=1}^{k-1} s_{[j]} + \frac{1}{k}s_y\right\} - \left(\frac{1}{k}\sum_{j=1}^{k-1} s_{[j]} + \frac{1}{k}s_y\right), \\
&= \max\left\{\max_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)} \setminus \mathcal{Y}_y^{(k)}} \left\{1 + \frac{1}{k}\sum_{j \in \bar{\mathbf{y}}} s_j\right\}, \max_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \left\{\frac{1}{k}\sum_{j \in \bar{\mathbf{y}}} s_j\right\}\right\} - \max_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \left\{\frac{1}{k}\sum_{j \in \bar{\mathbf{y}}} s_j\right\}, \\
&= \max_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}} \left\{\Delta_k(\bar{\mathbf{y}}, y) + \frac{1}{k}\sum_{j \in \bar{\mathbf{y}}} s_j\right\} - \max_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \left\{\frac{1}{k}\sum_{j \in \bar{\mathbf{y}}} s_j\right\}.
\end{aligned} \tag{A.3}$$

□

A.1.2 Point-wise Convergence

Lemma 1. *Let $n \geq 2$ and $\mathbf{e} \in \mathbb{R}^n$. Assume that the largest element of $\mathbf{e}$ is greater than its second largest element: $e_{[1]} > e_{[2]}$. Then $\lim_{\substack{\tau \to 0 \\ \tau > 0}} \tau \log\left(\sum_{i=1}^{n} \exp(e_i/\tau)\right) = e_{[1]}$.*

Proof. For simplicity of notation, and without loss of generality, we suppose that the elements of $\mathbf{e}$ are sorted in descending order. Then for $i \in \{2, ..n\}$, we have

$e_i - e_1 \leq e_2 - e_1 < 0$ by assumption, and thus $\forall i \in \{2, ..n\}$, $\lim_{\substack{\tau \to 0 \\ \tau > 0}} \exp((e_i - e_1)/\tau) = 0$. Therefore:

$$\lim_{\substack{\tau \to 0 \\ \tau > 0}} \sum_{i=1}^{n} \exp((e_i - e_1)/\tau) = \sum_{i=1}^{n} \lim_{\substack{\tau \to 0 \\ \tau > 0}} \exp((e_i - e_1)/\tau) = 1. \tag{A.4}$$

And thus:

$$\lim_{\substack{\tau \to 0 \\ \tau > 0}} \tau \log \left(\sum_{i=1}^{n} \exp((e_i - e_1)/\tau) \right) = 0. \tag{A.5}$$

The result follows by noting that:

$$\tau \log \left(\sum_{i=1}^{n} \exp(e_i/\tau) \right) = e_1 + \tau \log \left(\sum_{i=1}^{n} \exp((e_i - e_1)/\tau) \right). \tag{A.6}$$

□

Proposition 10. *Assume that* $s_{[k-1]} > s_{[k]}$ *and that* $s_{[k]} > s_{[k+1]}$ *or* $\frac{1}{k} s_y > 1 + \frac{1}{k} s_{[k]}$. *Then* $\lim_{\substack{\tau \to 0 \\ \tau > 0}} L_{k,\tau}(\mathbf{s}, y) = l_k(\mathbf{s}, y)$.

Proof. From $s_{[k]} > s_{[k+1]}$ or $\frac{1}{k} s_y > 1 + \frac{1}{k} s_{[k]}$, one can see that $\max_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}} \left\{ \Delta_k(\bar{\mathbf{y}}, y) + \frac{1}{k} \sum_{j \in \bar{\mathbf{y}}} s_j \right\}$ is a strict maximum. Similarly, from $s_{[k-1]} > s_{[k]}$, we can see that $\max_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \left\{ \frac{1}{k} \sum_{j \in \bar{\mathbf{y}}} s_j \right\}$ is a strict maximum. Since $L_{k,\tau}$ can be written as:

$$\begin{aligned} L_{k,\tau}(\mathbf{s}, y) = \tau \log \left[\sum_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}} \exp \left(\left(\Delta_k(\bar{\mathbf{y}}, y) + \frac{1}{k} \sum_{j \in \bar{\mathbf{y}}} s_j \right) / \tau \right) \right] \\ - \tau \log \left[\sum_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \exp \left(\left(\frac{1}{k} \sum_{j \in \bar{\mathbf{y}}} s_j \right) / \tau \right) \right], \end{aligned} \tag{A.7}$$

the result follows by two applications of Lemma 1. □

A.1.3 Bound on Non-Smooth Function

Proposition 11. $L_{k,\tau}$ *is an upper bound on* l_k *if and only if* $k = 1$.

Proof. Suppose $k = 1$. Let $\mathbf{s} \in \mathbb{R}^n$ and $y \in \mathcal{Y}$. We introduce $y^* = \operatorname{argmax}_{\bar{y} \in \mathcal{Y}} \{\Delta_1(\bar{y}, y) + s_{\bar{y}}\}$. Then we have:

$$\begin{aligned} l_1(\mathbf{s}, y) &= \Delta_1(y^*, y) + s_{y^*} - s_y, \\ &= \tau \log(\exp((\Delta_1(y^*, y) + s_{y^*})/\tau) - \tau \log \exp(s_y/\tau), \\ &\leq \tau \log(\sum_{\bar{y} \in \mathcal{Y}} \exp((\Delta_1(\bar{y}, y) + s_{\bar{y}})/\tau) - \tau \log \exp(s_y/\tau) = L_{1,\tau}(\mathbf{s}, y). \end{aligned} \tag{A.8}$$

Now suppose $k \geq 2$. We construct an example $(\mathbf{s}, y)$ such that $L_{k,\tau}(\mathbf{s}, y) < l_k(\mathbf{s}, y)$. For simplicity, we set $y = 1$. Then let $s_1 = \alpha$, $s_i = \beta$ for $i \in \{2, ..., k+1\}$ and $s_i = -\infty$ for $i \in \{k+2, ..., n\}$. The variables α and β are our degrees of freedom to construct the example. Assuming infinite values simplifies the analysis, and by continuity of $L_{k,\tau}$ and l_k, the proof will hold for real values sufficiently small. We further assume that $1 + \frac{1}{k}(\beta - \alpha) > 0$. Then can write $l_k(\mathbf{s}, y)$ as:

$$l_k(\mathbf{s}, y) = 1 + \frac{1}{k}(\beta - \alpha). \tag{A.9}$$

Exploiting the fact that $\exp(s_i/\tau) = 0$ for $i \geq k+2$, we have:

$$\sum_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)} \setminus \mathcal{Y}_y^{(k)}} \prod_{j \in \bar{\mathbf{y}}} \exp((1 + s_j)/k\tau) = \exp\left(\frac{1+\beta}{\tau}\right), \tag{A.10}$$

And:

$$\sum_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \exp\left(\left(\frac{1}{k}\sum_{j \in \bar{\mathbf{y}}} s_j\right)/\tau\right) = k \exp\left(\frac{\alpha + (k-1)\beta}{k\tau}\right). \tag{A.11}$$

This allows us to write $L_{k,\tau}$ as:

$$\begin{aligned}
L_{k,\tau}(\mathbf{s}, y) &= \tau \log\left(k \exp\left(\frac{\alpha + (k-1)\beta}{k\tau}\right) + \exp\left(\frac{1+\beta}{\tau}\right)\right) \\
&\quad - \tau \log\left(k \exp\left(\frac{\alpha + (k-1)\beta}{k\tau}\right)\right), \\
&= \tau \log\left(1 + \frac{\exp\left(\frac{1+\beta}{\tau}\right)}{k \exp\left(\frac{\alpha+(k-1)\beta}{k\tau}\right)}\right), \\
&= \tau \log\left(1 + \frac{\exp\left(\frac{1}{\tau}\right)}{k \exp\left(\frac{\alpha-\beta}{k\tau}\right)}\right), \\
&= \tau \log\left(1 + \frac{1}{k}\exp\left(\frac{1}{\tau}(1 + \frac{1}{k}(\beta - \alpha))\right)\right).
\end{aligned} \tag{A.12}$$

We introduce $x = 1 + \frac{1}{k}(\beta - \alpha)$. Then we have:

$$L_{k,\tau}(\mathbf{s}, y) = \tau \log\left(1 + \frac{1}{k}\exp\left(\frac{x}{\tau}\right)\right), \tag{A.13}$$

And:

$$l_k(\mathbf{s}, y) = x. \tag{A.14}$$

For any value $x > 0$, we can find $(\alpha, \beta) \in \mathbb{R}^2$ such that $x = 1 + \frac{1}{k}(\beta - \alpha)$ and that all our hypotheses are verified. Consequently, we only have to prove that there exists $x > 0$ such that:

$$\Delta(x) \triangleq \tau \log\left(1 + \frac{1}{k}\exp\left(\frac{x}{\tau}\right)\right) - x < 0. \tag{A.15}$$

We show that $\lim_{x\to\infty} \Delta(x) < 0$, which will conclude the proof by continuity of Δ.

$$\begin{aligned}
\Delta(x) &= \tau \log\left(1 + \frac{1}{k}\exp\left(\frac{x}{\tau}\right)\right) - x, \\
&= \tau \log\left(1 + \frac{1}{k}\exp\left(\frac{x}{\tau}\right)\right) - \tau \log(\exp(\frac{x}{\tau})), \\
&= \tau \log\left(\exp\left(\frac{-x}{\tau}\right) + \frac{1}{k}\right) \xrightarrow[x\to\infty]{} \tau \log(\frac{1}{k}) < 0 \quad \text{since } k \geq 2.
\end{aligned} \tag{A.16}$$

□

A.1.4 Bound on Prediction Loss

Lemma 2. *Let $(p, q) \in \mathbb{N}^2$ such that $p \leq q - 1$ and $q \geq 1$. Then $\binom{q}{p} \leq q\binom{q}{p+1}$.*

Proof.

$$\begin{aligned}
\frac{\binom{q}{p}}{\binom{q}{p+1}} &= \frac{(q-p-1)!(p+1)!}{(q-p)!p!}, \\
&= \frac{(p+1)}{q-p}.
\end{aligned} \tag{A.17}$$

This is a monotonically increasing function of $p \leq q - 1$, therefore it is upper bounded by its maximal value at $p = q - 1$:

$$\frac{\binom{q}{p}}{\binom{q}{p+1}} = \frac{(p+1)}{q-p} \leq q. \tag{A.18}$$

□

Lemma 3. *Assume that $y \notin P_k(\mathbf{s})$. Then we have:*

$$\frac{1}{k}\sum_{\bar{\mathbf{y}}\in\mathcal{Y}_y^{(k)}} \exp\left(\sum_{j\in\bar{\mathbf{y}}} \frac{s_j}{k\tau}\right) \leq \sum_{\bar{\mathbf{y}}\in\mathcal{Y}^{(k)}\setminus\mathcal{Y}_y^{(k)}} \exp\left(\sum_{j\in\bar{\mathbf{y}}} \frac{s_j}{k\tau}\right). \tag{A.19}$$

Proof. Let $j \in [\![0, k-1]\!]$. We introduce the random variable U_j, whose probability distribution is uniform over the set $\mathcal{U}_j \triangleq \{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)} : \bar{\mathbf{y}} \cap P_k(\mathbf{s}) = j\}$. Then V_j is the random variable such that $V_j|U_j$ replaces y from U_j with a value drawn uniformly from $P_k(\mathbf{s})$. We denote by $\mathcal{V}_j$ the set of values taken by V_j with non-zero probability. Since V_j replaces the ground truth score by one of the values of $P_k(\mathbf{s})$, it can be seen that:

$$\mathcal{V}_j = \{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)} \backslash \mathcal{Y}_y^{(k)} : \bar{\mathbf{y}} \cap P_k(\mathbf{s}) = j+1\}. \tag{A.20}$$

Furthermore, we introduce the scoring function $f : \bar{\mathbf{y}} \in \mathcal{Y}^{(k)} \mapsto \exp(\frac{1}{k\tau} \sum_{j \in \bar{\mathbf{y}}} s_j)$. Since $P_k(\mathbf{s})$ is the set of the k largest scores and $y \notin P_k(\mathbf{s})$, we have that:

$$f(V_j|U_j) \geq f(U_j) \qquad \text{with probability 1.} \tag{A.21}$$

Therefore we also have that:

$$\mathbb{E}_{V_j|U_j} f(V_j) \geq f(U_j) \qquad \text{with probability 1.} \tag{A.22}$$

This finally gives us:

$$\begin{aligned} \mathbb{E}_{U_j} \mathbb{E}_{V_j|U_j} f(V_j) &\geq \mathbb{E}_{U_j} f(U_j), \\ \mathbb{E}_{V_j} f(V_j) &\geq \mathbb{E}_{U_j} f(U_j). \end{aligned} \tag{A.23}$$

Making the (uniform) probabilities explicit, we obtain:

$$\begin{aligned} \frac{1}{|\mathcal{V}_j|} \sum_{\mathbf{v} \in \mathcal{V}_j} f(\mathbf{v}) &\geq \frac{1}{|\mathcal{U}_j|} \sum_{\mathbf{u} \in \mathcal{U}_j} f(\mathbf{u}), \\ \frac{|\mathcal{U}_j|}{|\mathcal{V}_j|} \sum_{\mathbf{v} \in \mathcal{V}_j} f(\mathbf{v}) &\geq \sum_{\mathbf{u} \in \mathcal{U}_j} f(\mathbf{u}). \end{aligned} \tag{A.24}$$

To derive the set cardinalities, we rewrite $\mathcal{U}_j$ and $\mathcal{V}_j$ as:

$$\begin{aligned} \mathcal{U}_j &= \{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)} : \bar{\mathbf{y}} \cap P_k(\mathbf{s}) = j\} = \{y\} \times P_k(\mathbf{s})^{(j)} \times (\mathcal{Y} \backslash (\{y\} \cup P_k(\mathbf{s}))^{(k-j-1)}, \\ \mathcal{V}_j &= \{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)} \backslash \mathcal{Y}_y^{(k)} : \bar{\mathbf{y}} \cap P_k(\mathbf{s}) = j+1\} = P_k(\mathbf{s})^{(j+1)} \times (\mathcal{Y} \backslash (\{y\} \cup P_k(\mathbf{s}))^{(k-j-1)}. \end{aligned} \tag{A.25}$$

Therefore we have that:

$$\begin{aligned} |\mathcal{U}_j| &= \left| \{y\} \times P_k(\mathbf{s})^{(j)} \times (\mathcal{Y} \backslash (\{y\} \cup P_k(\mathbf{s}))^{(k-j-1)} \right|, \\ &= \binom{k}{j} \binom{n-k-1}{k-j-1}, \end{aligned} \tag{A.26}$$

And:

$$\begin{aligned} |\mathcal{V}_j| &= \left| P_k(\mathbf{s})^{(j+1)} \times (\mathcal{Y} \backslash (\{y\} \cup P_k(\mathbf{s}))^{(k-j-1)} \right|, \\ &= \binom{k}{j+1} \binom{n-k-1}{k-j-1}. \end{aligned} \tag{A.27}$$

Therefore:

$$\frac{|\mathcal{U}_j|}{|\mathcal{V}_j|} = \frac{\binom{k}{j}\binom{n-k-1}{k-j-1}}{\binom{k}{j+1}\binom{n-k-1}{k-j-1}} = \frac{\binom{k}{j}}{\binom{k}{j+1}} \leq k \quad \text{by Lemma 2.} \tag{A.28}$$

Combining with equation (A.24), we obtain:

$$k \sum_{\mathbf{v} \in \mathcal{V}_j} f(\mathbf{v}) \geq \sum_{\mathbf{u} \in \mathcal{U}_j} f(\mathbf{u}). \tag{A.29}$$

We sum over $j \in [\![0, k-1]\!]$, which yields:

$$k \sum_{j=0}^{k-1} \sum_{\mathbf{v} \in \mathcal{V}_j} f(\mathbf{v}) \geq \sum_{j=0}^{k-1} \sum_{\mathbf{u} \in \mathcal{U}_j} f(\mathbf{u}). \tag{A.30}$$

Finally, we note that $\{\mathcal{U}_j\}_{0 \leq j \leq k-1}$ and $\{\mathcal{V}_j\}_{0 \leq j \leq k-1}$ are respective partitions of $\mathcal{Y}_y^{(k)}$ and $\mathcal{Y}^{(k)} \backslash \mathcal{Y}_y^{(k)}$, which gives us the final result:

$$k \sum_{\mathbf{v} \in \mathcal{Y}^{(k)} \backslash \mathcal{Y}_y^{(k)}} f(\mathbf{v}) \geq \sum_{\mathbf{u} \in \mathcal{Y}_y^{(k)}} f(\mathbf{u}). \tag{A.31}$$

□

Proposition 12. *$L_{k,\tau}$ is, up to a scaling factor, an upper bound on the prediction loss Λ_k:*

$$L_{k,\tau}(\mathbf{s}, y) \geq (1 - \tau \log(k)) \Lambda_k(\mathbf{s}, y). \tag{A.32}$$

Proof. Suppose that $\Lambda_k(\mathbf{s}, y) = 0$. Then the inequality is trivial because $L_{k,\tau}(\mathbf{s}, y) \geq 0$. We now assume that $\Lambda_k(\mathbf{s}, y) = 1$. Then there exist at least k higher scores than s_y. To simplify indexing, we introduce $\mathcal{Z}_y^{(k)} = \mathcal{Y}^{(k)} \backslash \mathcal{Y}_y^{(k)}$ and $\mathcal{T}_k$ the set of k labels corresponding to the k-largest scores. By assumption, $y \notin \mathcal{T}_k$ since y is misclassified. We then write:

$$\sum_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}} \exp\left(\Delta(\bar{\mathbf{y}}, y)/\tau\right) \prod_{j \in \bar{\mathbf{y}}} u_j = \exp\left(1/\tau\right) \sum_{\bar{\mathbf{y}} \in \mathcal{Z}_y^{(k)}} \prod_{j \in \bar{\mathbf{y}}} u_j + \sum_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \prod_{j \in \bar{\mathbf{y}}} u_j. \tag{A.33}$$

Thanks to Lemma 3, we have:

$$\sum_{\bar{\mathbf{y}} \in \mathcal{Z}_y^{(k)}} \prod_{j \in \bar{\mathbf{y}}} u_j \geq \frac{1}{k} \sum_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \prod_{j \in \bar{\mathbf{y}}} u_j. \tag{A.34}$$

Injecting this back into (A.33):

$$\sum_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}} \exp\left(\Delta(\bar{\mathbf{y}}, y)/\tau\right) \prod_{j \in \bar{\mathbf{y}}} u_j \geq (1 + \frac{1}{k} \exp\left(1/\tau\right)) \sum_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \prod_{j \in \bar{\mathbf{y}}} u_j, \tag{A.35}$$

And back to the original loss:

$$\begin{aligned} L_{k,\tau}(\mathbf{s}, y) &\geq \tau \log \left[(1 + \frac{1}{k} \exp\left(1/\tau\right)) \sum_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \prod_{j \in \bar{\mathbf{y}}} u_j \right] - \tau \log \left[\sum_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \prod_{j \in \bar{\mathbf{y}}} u_j \right], \\ &= \tau \log(1 + \frac{1}{k} \exp\left(1/\tau\right)), \\ &\geq \tau \log(\frac{1}{k} \exp\left(1/\tau\right)), \\ &= \tau \log(\frac{1}{k}) + 1, \\ &= 1 - \tau \log(k). \end{aligned} \tag{A.36}$$

□

A.2 Algorithms: Properties & Performance

A.2.1 Time Complexity

Lemma 4. *Let P and Q be two polynomials of degree p and q. The time complexity of obtaining the first r coefficients of PQ is $\mathcal{O}(\min\{r, p\} \min\{r, q\})$.*

Proof. The multiplication of two polynomials can be written as the convolution of their coefficients, which can be truncated at degree r for each polynomial. □

Proposition 13. *The time complexity of Algorithm 1 is $\mathcal{O}(kn)$.*

Proof. Let $N = \log_2(n)$, or equivalently $n = 2^N$. With the divide-and-conquer algorithm, the complexity of computing the k first coefficients of P can be written as:

$$T(k, n) = 2T(k, \frac{n}{2}) + \min\{k, n\}^2. \tag{A.37}$$

Indeed we decompose $P = Q_1 Q_2$, with each Q_i of degree $n/2$, and for these we compute their k first coefficients in $T(\frac{n}{2})$. Then given the k first coefficients of Q_1 and Q_2, the k first coefficients of P are computed in $\mathcal{O}(\min\{k,n\}^2)$ by Lemma 4. Then we can write:

$$\begin{aligned} T(k,n) &= 2T\Big(k,\frac{n}{2}\Big) + \min\{k,n\}^2, \\ 2T\Big(k,\frac{n}{2}\Big) &= 4T\Big(k,\frac{n}{4}\Big) + 2\min\Big\{k,\frac{n}{2}\Big\}^2, \\ &\dots \\ 2^{N-1}T\Big(k,\frac{n}{2^{N-1}}\Big) &= \underbrace{2^N T(k,1)}_{2^N\mathcal{O}(1)=\mathcal{O}(n)} + 2^{N-1}\min\Big\{k,\frac{n}{2^{N-1}}\Big\}^2. \end{aligned} \tag{A.38}$$

By summing these terms, we obtain $T(k,n) = 2^N T(k,1) + \sum_{j=0}^{N-1} 2^j \min\Big\{k,\frac{n}{2^j}\Big\}^2$. Let $n_0 \in \mathbb{N}$ such that $\frac{n}{2^{n_0+1}} < k \leq \frac{n}{2^{n_0}}$. In loose notation, we have $k\frac{2^{n_0}}{n} = \mathcal{O}(1)$. Then we can write:

$$\begin{aligned} \sum_{j=0}^{N-1} 2^j \min\Big\{k,\frac{n}{2^j}\Big\}^2 &= \sum_{j=0}^{n_0} 2^j \min\Big\{k,\frac{n}{2^j}\Big\}^2 + \sum_{j=n_0+1}^{N-1} 2^j \min\Big\{k,\frac{n}{2^j}\Big\}^2, \\ &= \sum_{j=0}^{n_0} 2^j k^2 + \sum_{j=n_0+1}^{N-1} 2^j \Big(\frac{n}{2^j}\Big)^2, \\ &= (2^{n_0+1}-1)k^2 + n^2(2^{-n_0-1} - 2^{-N}), \\ &= \mathcal{O}(kn). \end{aligned} \tag{A.39}$$

Thus finally:

$$\begin{aligned} T(k,n) &= 2^N T(k,1) + \sum_{j=0}^{N-1} 2^j \min\Big\{k,\frac{n}{2^j}\Big\}^2, \\ &= \mathcal{O}(n) + \mathcal{O}(kn), \\ &= \mathcal{O}(kn). \end{aligned} \tag{A.40}$$

□

A.2.2 Numerical Stability

Forward Pass

In order to ensure numerical stability of the computation, we maintain all computations in the log space: for a multiplication $\exp(x_1)\exp(x_2)$, we actually compute

and store $x_1 + x_2$; for an addition $\exp(x_1) + \exp(x_2)$ we use the "log-sum-exp" trick: we compute $m = \max\{x_1, x_2\}$, and store $m + \log(\exp(x_1 - m) + \exp(x_2 - m))$, which guarantees stability of the result. These two operations suffice to describe the forward pass.

Backward Pass

Observation 2. *The backward recursion of Algorithm 2 is unstable when $e_i \gg 1$ and $e_i \gg \max_{p \neq i}\{e_p\}$.*

Sketch of Proof. To see that, assume that when we compute $(\sum_{p=1}^{n} e_p) - e_i$, we make a numerical error in the order of ϵ (e.g $\epsilon \simeq 10^{-5}$ for single floating point precision). With the numerical errors, we obtain approximate $\hat{\delta}$ as follows:

$$\begin{aligned}
&\hat{\delta}_{1,i} = 1, \\
&\hat{\delta}_{2,i} = \sigma_1(\mathbf{e}) - e_i\hat{\delta}_{1,i} = \sum_{p=1}^{n} e_p - e_i = \delta_{2,i} + \mathcal{O}(\epsilon), \\
&\hat{\delta}_{3,i} = \sigma_2(\mathbf{e}) - e_i\hat{\delta}_{2,i} = \sigma_2(\mathbf{e}) - e_i(\delta_{2,i} + \mathcal{O}(\epsilon)) = \delta_{3,i} + \mathcal{O}(e_i\epsilon)), \\
&\quad ... \\
&\hat{\delta}_{k,i} = \sigma_{k-1}(\mathbf{e}) - e_i\hat{\delta}_{k-1,i} = ... = \delta_{k,i} + \mathcal{O}(e_i^{k-1}\epsilon)).
\end{aligned} \tag{A.41}$$

Since $e_i \gg 1$, we quickly obtain unstable results. □

Definition 1. *For $p \in \{0, ..., n-k\}$, we define the p-th order approximation to the gradient as:*

$$\tilde{\delta}_{k,i}^{(p)} \triangleq \sum_{j=0}^{p} (-1)^j \frac{\sigma_{k+j}(\mathbf{e})}{e_i^j}. \tag{A.42}$$

Proposition 14. *If we approximate the gradient by its p-th order approximation as defined in equation (A.42), the absolute error is:*

$$\left|\delta_{k,i} - \tilde{\delta}_{k,i}^{(p)}\right| = \frac{\sigma_{k+p}(\mathbf{e}_{\setminus i})}{e_i^{p+1}}. \tag{A.43}$$

Proof. We remind equation (2.18), which gives a recursive relationship for the gradients:

$$\delta_{j,i} = \sigma_{j-1}(\mathbf{e}) - e_i\delta_{j-1,i}.$$

This can be re-written as:

$$\delta_{j-1,i} = \frac{1}{e_i} \left(\sigma_{j-1}(\mathbf{e}) - \delta_{j,i} \right). \tag{A.44}$$

We write $\sigma_{k+p}(\mathbf{e}_{\setminus i}) = \delta_{k+p+1,i}$, and the result follows by repeated applications of equation (A.44) for $j \in \{k+1, k+2, ..., k+p+1\}$. □

Intuition. We have seen in Observation 2 that the recursion tends to be unstable for $\delta_{j,i}$ when e_i is among the largest elements. When that is the case, the ratio $\frac{\sigma_{k+p}(\mathbf{e}_{\setminus i})}{e_i^{p+1}}$ decreases quickly with p. This has two consequences: (i) the sum of equation (A.42) is stable to compute because the summands have different orders of magnitude and (ii) the error becomes small. Unfortunately, it is difficult to upper-bound the error of equation (A.43) by a quantity that is both measurable at runtime (without expensive computations) and small enough to be informative. Therefore the approximation error is not controlled at runtime. In practice, we detect the instability of $\delta_{k,i}$: numerical issues arise if subtracted terms have a very small relative difference. For those unstable elements we use the p-th order approximation (to choose the value of p, a good rule of thumb is $p \simeq 0.2k$). We have empirically found out that this heuristic works well in practice. Note that this changes the complexity of the forward pass to $\mathcal{O}((k+p)n)$ since we need p additional coefficients during the backward. If $p \simeq 0.2k$, this increases the running time of the forward pass by 20%, which is a moderate impact.

A.2.3 A Performance Comparison with the Summation Algorithm

Summation Algorithm

The Summation Algorithm (SA) is an alternative to the Divide-and-Conquer (DC) algorithm for the evaluation of the elementary symmetric polynomials. It is described for instance in (Jiang et al., 2016). The algorithm can be summarized as follows:

Algorithm 6 *Summation Algorithm*

Require: $\mathbf{e} \in \mathbb{R}^n$, $k \in \mathbb{N}^*$
$\sigma_{0,i} \leftarrow 1$ for $1 \leq i \leq n$ ▷ $\sigma_{j,i} = \sigma_j(e_1, \dots, e_i)$
$\sigma_{j,i} \leftarrow 0$ for $i < j$ ▷ Do not define values for $i < j$ (meaningless)
$\sigma_{1,1} \leftarrow e_1$ ▷ Initialize recursion
for $i \in [\![2, n]\!]$ **do**
    $m \leftarrow \max\{1, i + k - n\}$
    $M \leftarrow \min\{i, k\}$
    for $i \in [\![m, M]\!]$ **do**
        $\sigma_{j,i} \leftarrow \sigma_{j,i-1} + e_i \sigma_{j-1,i-1}$
    end for
end for
return $\sigma_{k,n}$

Implementation. Note that the inner loop can be parallelized, but the outer one is essentially sequential. In our implementation for speed comparisons, the inner loop is parallelized and a buffer is pre-allocated for the $\sigma_{j,i}$.

Speed

We compare the execution time of the DC and SA algorithms on a GPU (Nvidia Titan Xp). We use the following parameters: $k = 5$, a batch size of 256 and a varying value of n. The following timings are given in seconds, and are computed as the average of 50 runs. In Table A.1, we compare the speed of Summation and DC for the evaluation of the forward pass. In Table A.2, we compare the speed of the evaluation of the backward pass using Automatic Differentiation (AD) and our Custom Algorithm (CA) (see Algorithm 2).

Table A.1: *Execution time (s) of the forward pass. The Divide and Conquer (DC) algorithm offers nearly logarithmic scaling with n in practice, thanks to its parallelization. In contrast, the runtime of the Summation Algorithm (SA) scales linearly with n.*

n	100	1,000	10,000	100,000
SA	0.006	0.062	0.627	6.258
DC	0.011	0.018	0.024	0.146

We remind that both algorithms have a time complexity of $\mathcal{O}(kn)$. SA provides little parallelization (the parallelizable inner loop is small for $k \ll n$), which is reflected in the runtimes. On the other hand, DC is a recursive algorithm with

$\mathcal{O}(\log(n))$ levels of recursion, and all operations are parallelized at each level of the recursion. This allows DC to have near-logarithmic effective scaling with n, at least in the range $\{100 - 10,000\}$.

Table A.2: *Execution time (s) of the backward pass. Our Custom Backward (CB) is faster than Automatic Differentiation (AD).*

n	100	1,000	10,000	100,000
DC (AD)	0.093	0.139	0.194	0.287
DC (CB)	0.007	0.006	0.020	0.171

These runtimes demonstrate the advantage of using Algorithm 2 instead of automatic differentiation. In particular, we see that in the use case of ImageNet ($n = 1,000$), the backward computation changes from being 8x slower than the forward pass to being 3x faster.

Stability

We now investigate the numerical stability of the algorithms. Here we only analyze the numerical stability, and not the precision of the algorithm. We point out that compensation algorithms are useful to improve the precision of SA but not its stability. Therefore they are not considered in this discussion.

Jiang et al. (2016) mention that SA is a stable algorithm, under the assumption that no overflow or underflow is encountered. However this assumption is not verified in our use case, as we demonstrate below. We consider that the algorithm is stable if no overflow occurs in the algorithm (underflows are not an issue for our use cases). We stress out that numerical stability is critical for our machine learning context: if an overflow occurs, the weights of the learning model inevitably diverge to infinite values.

To test numerical stability in a representative setting of our use cases, we take a random mini-batch of 128 images from the ImageNet data set and forward it through a pre-trained ResNet-18 to obtain a vector of scores per sample. Then we use the scores as an input to the SA and DC algorithms, for various values of the temperature parameter τ. We compare the algorithms with single (S) and double (D) floating point precision.

Table A.3: *Stability on forward pass. A setting is considered stable if no overflow has occurred.*

τ	10^1	10^0	10^{-1}	10^{-2}	10^{-3}	10^{-4}
SA (S)	✓	✓	✗	✗	✗	✗
SA (D)	✓	✓	✓	✗	✗	✗
DC log (S)	✓	✓	✓	✓	✓	✓
DC log (D)	✓	✓	✓	✓	✓	✓

By operating in the log-space, DC is significantly more stable than SA. In this experimental setting, DC log is stable in single floating point precision until $\tau = 10^{-36}$.

A.3 Top-k Prediction: Marginalization with the Elementary Symmetric Polynomials

We consider the probability of label i being part of the final top-k prediction. To that end, we marginalize over all k-tuples that contain i as one of their element. Then the probability of selecting label i for the top-k prediction can be written as:

$$p_i^{(k)} \propto \sum_{\bar{\mathbf{y}} \in \mathcal{Y}_i^{(k)}} \exp(\sum_{j \in \bar{\mathbf{y}}} s_j). \tag{A.45}$$

Proposition 15. *The unnormalized probability can be computed as:*

$$p_i^{(k)} \propto \frac{d \log \sigma_i(\exp(\mathbf{s}))}{ds_i}. \tag{A.46}$$

Proof.

$$\begin{aligned} p_i^{(k)} &\propto \exp(s_i)\sigma_{k-1}(\exp(\mathbf{s}_{\setminus i})), \\ &= \exp(s_i)\frac{d\sigma_i(\exp(\mathbf{s}))}{d\exp(s_i)}, \\ &= \frac{d\sigma_i(\exp(\mathbf{s}))}{ds_i}. \end{aligned} \tag{A.47}$$

Finally we can rescale the unnormalized probability $p_i^{(k)}$ by $\sigma_k(\exp(\mathbf{s}))$ since the latter quantity is independent of i. We obtain:

$$\hat{p_i}^{(k)} \propto \frac{1}{\sigma_k(\exp(\mathbf{s}))}\frac{d\sigma_i(\exp(\mathbf{s}))}{ds_i} = \frac{d \log \sigma_i(\exp(\mathbf{s}))}{ds_i}. \tag{A.48}$$

□

NB. We prefer to use $\frac{d \log \sigma_i(\exp(\mathbf{s}))}{ds_i}$ rather than $\frac{d\sigma_i(\exp(\mathbf{s}))}{ds_i}$ for stability reasons. Once the unnormalized probabilities are computed, they can be normalized by simply dividing by their sum.

A.4 Hyper-Parameters & Experimental Details

A.4.1 The Temperature Parameter

In this section, we discuss the choice of the temperature parameter. Note that such insights are not necessarily confined to a top-k minimization: we believe that these ideas generalize to any loss that is smoothed with a temperature parameter.

Optimization and Learning

When the temperature τ has a low value, propositions 11 and 12 suggest that $L_{k,\tau}$ is a sound learning objective. However, as shown in Figure 2.2a, optimization is difficult and can fail in practice. Conversely, optimization with a high value of the temperature is easy, but uninformative about the learning: then $L_{k,\tau}$ is not representative of the task loss we wish to learn.

In other words, there is a trade-off between the ease of the optimization and the quality of the surrogate loss in terms of learning. Therefore, it makes sense to use a low temperature that still permits satisfactory optimization.

Illustration on CIFAR-100

In Figure 2.2a, we have provided the plots of the training objective to illustrate the speed of convergence. In Table A.4, we give the training and validation accuracies to show the influence of the temperature:

Table A.4: *Influence of the temperature parameter on the training accuracy and testing accuracy.*

Temperature	Training Accuracy (%)	Testing Accuracy (%)
0	10.01	10.38
10^{-3}	17.40	18.19
10^{-2}	98.95	91.35
10^{-1}	99.73	91.70
10^{0}	99.78	91.52
10^{1}	99.62	90.92
10^{2}	99.42	90.46

To Anneal or Not To Anneal

The choice of temperature parameter can affect the scale of the loss function. In order to preserve a sensible trade-off between regularizer and loss, it is important to adjust the regularization hyper-parameter(s) accordingly (the value of the quadratic regularization for instance). Similarly, the energy landscape may vary significantly for a different value of the temperature, and the learning rate may need to be adapted too.

Continuation methods usually rely on an annealing scheme to gradually improve the quality of the approximation. For this work, we have found that such an approach required heavy engineering and did not provide substantial improvement in our experiments. Indeed, we have mentioned that other hyper-parameters depend on the temperature, thus these need to be adapted dynamically too. This requires sensitive heuristics. Furthermore, we empirically find that setting the temperature to an appropriate fixed value yields the same performance as careful fine-tuning of a pre-trained network with temperature annealing.

Practical Methodology

We summarize the methodology that reflects the previous insights and that we have found to work well during our experimental investigation. First, the temperature hyper-parameter is set to a low fixed value that allows for the model to learn on the training data set. Then other hyper-parameters, such as quadratic regularization and learning rate are adapted as usual by cross-validation on the validation set.

We believe that the optimal value of the temperature is mostly independent of the architecture of the neural network, but is greatly influenced by the values of k and n (see how these impact the number of summands involved in $L_{k,\tau}$, and therefore its scale).

A.4.2 The Margin

Relationship with Squared Norm Regularization

In this subsection, we establish the relationship between hyper-parameters of the margin and of the regularization with a squared norm. Typically the regularizing norm is the Frobenius norm in deep learning, but the following results will follow for any norm $\|\cdot\|$. Although we prove the result for our top-k loss, we also point out that these results easily generalize to any linear latent structural SVM.

First, we make explicit the role of α in l_k with an overload of notation:

$$l_k(\mathbf{s}, y, \alpha) = \max\left\{\left(\mathbf{s}_{\setminus y} + \alpha\mathbf{1}\right)_{[k]} - s_y, 0\right\}, \tag{A.49}$$

where α is a non-negative real number. Now consider the problem of learning a linear top-k SVM on a data set $(\mathbf{x}_i, y_i)_{1\leq i\leq N} \in \left(\mathbb{R}^d \times \{1, ..., n\}\right)^N$. We (hyper-)parameterize this problem by λ and α:

$$(P_{\lambda,\alpha}): \qquad \min_{\mathbf{w}\in\mathbb{R}^{d\times n}} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{N}\sum_{i=1}^{N} l_k(\mathbf{w}^T\mathbf{x}_i, y_i, \alpha). \tag{A.50}$$

Definition 2. *Let $\lambda_1, \lambda_2, \alpha_1, \alpha_2 \geq 0$. We say that (P_{λ_1,α_1}) and (P_{λ_2,α_2}) are equivalent if there exists $\gamma > 0, \nu \in \mathbb{R}$ such that:*

$$\mathbf{w} \in \mathbb{R}^{d\times n} \text{ is a solution of } (P_{\lambda_1,\alpha_1}) \iff (\gamma\mathbf{w}+\nu) \text{ is a solution of } (P_{\lambda_2,\alpha_2}) \tag{A.51}$$

Justification. This definition makes sense because for $\gamma > 0, \nu \in \mathbb{R}$, $(\gamma\mathbf{w}+\nu)$ has the same decision boundary as $\mathbf{w}$. In other words, equivalent problems yield equivalent classifiers.

Proposition 16. *Let $\lambda, \alpha \geq 0$.*

1. *If $\alpha > 0$ and $\lambda > 0$, then problem $(P_{\lambda,\alpha})$ is equivalent to problems $(P_{\alpha\lambda,1})$ and $(P_{1,\alpha\lambda})$.*

2. If $\alpha = 0$ or $\lambda = 0$, then problem $(P_{\lambda,\alpha})$ is equivalent to problem $(P_{0,0})$.

Proof. Let $\mathbf{w} \in \mathbb{R}^{d\times n}$. We introduce a constant $\beta > 0$. Then we can successively write:

$$
\begin{aligned}
&\mathbf{w} \text{ is a solution to } (P_{\lambda,\alpha}) \\
&\iff \mathbf{w} \text{ is a solution to} \\
&\quad \min_{\mathbf{w}\in\mathbb{R}^{d\times n}} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{N}\sum_{i=1}^{N} l_k(\mathbf{w}^T\mathbf{x}_i, y_i, \alpha), \\
&\iff \mathbf{w} \text{ is a solution to} \\
&\quad \min_{\mathbf{w}\in\mathbb{R}^{d\times n}} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{N}\sum_{i=1}^{N} \max\left\{\left(\mathbf{w}_{\setminus y}^T\mathbf{x}_i + \alpha\mathbf{1}\right)_{[k]} - \mathbf{w}_y^T\mathbf{x}_i, 0\right\}, \\
&\iff \mathbf{w} \text{ is a solution to} \\
&\quad \min_{\mathbf{w}\in\mathbb{R}^{d\times n}} \frac{\lambda}{2\beta}\|\mathbf{w}\|^2 + \frac{1}{N}\sum_{i=1}^{N} \max\left\{\left(\frac{1}{\beta}\mathbf{w}_{\setminus y}^T\mathbf{x}_i + \frac{\alpha}{\beta}\mathbf{1}\right)_{[k]} - \frac{1}{\beta}\mathbf{w}_y^T\mathbf{x}_i, 0\right\}, \qquad \text{(A.52)} \\
&\iff \mathbf{w} \text{ is a solution to} \\
&\quad \min_{\mathbf{w}\in\mathbb{R}^{d\times n}} \frac{\beta\lambda}{2}\|\frac{1}{\beta}\mathbf{w}\|^2 + \frac{1}{N}\sum_{i=1}^{N} \max\left\{\left(\frac{1}{\beta}\mathbf{w}_{\setminus y}^T\mathbf{x}_i + \frac{\alpha}{\beta}\mathbf{1}\right)_{[k]} - \frac{1}{\beta}\mathbf{w}_y^T\mathbf{x}_i, 0\right\}, \\
&\iff \frac{1}{\beta}\mathbf{w} \text{ is a solution to} \\
&\quad \min_{\mathbf{w}\in\mathbb{R}^{d\times n}} \frac{\beta\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{N}\sum_{i=1}^{N} \max\left\{\left(\mathbf{w}_{\setminus y}^T\mathbf{x}_i + \frac{\alpha}{\beta}\mathbf{1}\right)_{[k]} - \mathbf{w}_y^T\mathbf{x}_i, 0\right\}, \\
&\iff \frac{1}{\beta}\mathbf{w} \text{ is a solution to } (P_{\beta\lambda,\frac{\alpha}{\beta}}).
\end{aligned}
$$

This holds for any $\beta > 0$.

If $\alpha > 0$ and $\lambda > 0$, we show equivalence with $(P_{\alpha\lambda,1})$ by setting β to α and with $(P_{1,\alpha\lambda})$ by setting β to $\frac{1}{\lambda}$.

If $\alpha = 0$, then $\frac{\alpha}{\beta} = 0$ for any $\beta > 0$ and we can choose β as small as needed to make $\beta\lambda$ arbitrarily small.

If $\lambda = 0$, $\beta\lambda = 0$ for any $\beta > 0$ and we can choose β as large as needed to make $\frac{\alpha}{\beta}$ arbitrarily small.

Note that we do not need any hypothesis on the norm $\|\cdot\|$, the result makes only use of the positive homogeneity property. □

Consequence On Deep Networks. Proposition 16 shows that for a deep network trained with l_k, one can fix the value of α to 1, and treat the quadratic regularization of the last fully connected layer as an independent hyper-parameter. By doing this rather than tuning α, the loss keeps the same scale which may make it easier to find an appropriate learning rate.

When using the smooth loss, there is no direct equivalent to Proposition 16 because the log-sum-exp function is not positively homogeneous. However one can consider that with a low enough temperature, the above insight can still be used in practice.

Experiment on ImageNet

In this section, we provide experiments to qualitatively assess the importance of the margin by running experiments with a margin of either 0 or 1. The following results are obtained on our validation set, and do not make use of multiple crops.

Top-1 Error. As we have mentioned before, the case $(k, \tau, \alpha) = (1, 1, 0)$ corresponds exactly to Cross-Entropy. We compare this case against the same loss with a margin of 1: $(k, \tau, \alpha) = (1, 1, 1)$. We obtain the following results:

Margin	Top-1 Accuracy (%)
0	71.03
1	**71.15**

Table A.5: *Influence of the margin parameter on top-*1 *performance.*

Top-5 Error. We now compare $(k, \tau, \alpha) = (5, 0.1, 0)$ and $(k, \tau, \alpha) = (5, 0.1, 1)$:

Margin	Top-5 Accuracy (%)
0	89.12
1	**89.45**

Table A.6: *Influence of the margin parameter on top-*5 *performance.*

A.4.3 Supplementary Details

In the main paper, we report the average of the scores on CIFAR-100 for clarity purposes. Here, we also detail the standard deviation of the scores for completeness.

Table A.7: *Testing performance on CIFAR-100 with different levels of label noise. We indicate the mean and standard deviation (in parenthesis) for each score.*

Noise	Top-1 Accuracy (%)		Top-5 Accuracy (%)	
Level	CE	$L_{5,1}$	CE	$L_{5,1}$
0.0	**76.68** (0.38)	69.33 (0.27)	**94.34** (0.09)	94.29 (0.10)
0.2	68.20 (0.50)	**71.30** (0.79)	87.89 (0.08)	**90.59** (0.08)
0.4	61.18 (0.97)	**70.02** (0.40)	83.04 (0.38)	**87.39** (0.23)
0.6	52.50 (0.27)	**67.97** (0.51)	79.59 (0.36)	**83.86** (0.39)
0.8	35.53 (0.79)	**55.85** (0.80)	74.80 (0.15)	**79.32** (0.25)
1.0	14.06 (0.13)	**15.28** (0.39)	67.70 (0.16)	**72.93** (0.25)

B

Trusting SVMs for Piecewise Linear CNNs

Contents

B.1 Piecewise Linear Functions

Proof of Proposition (1) By the definition from (Melzer, 1986), we can write each function as the difference of two point-wise maxima of linear functions:

$$g(v) = \max_{j\in[m_+]}\{\mathbf{a}_i^\top \mathbf{v}\} - \max_{k\in[m_-]}\{\mathbf{b}_j^\top \mathbf{v}\}$$
$$\text{And } \forall i \in [n], g_i(u) = g_i^+(u) - g_i^-(u)$$

Where all the g_i^+, g_i^- are linear point-wise maxima of linear functions. Then:

$$\begin{aligned}
f(u) &= g([g_1(\mathbf{u}), \cdots, g_n(\mathbf{u})]^\top) \\
&= \max_{j\in[m_+]}\{\mathbf{a}_j^\top [g_1(\mathbf{u}), \cdots, g_n(\mathbf{u})]^\top\} - \max_{k\in[m_-]}\{\mathbf{b}_k^\top [g_1(\mathbf{u}), \cdots, g_n(\mathbf{u})]^\top\} \\
&= \max_{j\in[m_+]}\left\{\sum_{i=1}^n a_{j,i} g_i(\mathbf{u})\right\} - \max_{k\in[m_-]}\left\{\sum_{i=1}^n b_{k,i} g_i(\mathbf{u})\right\} \\
&= \max_{j\in[m_+]}\left\{\sum_{i=1}^n a_{j,i} g_i^+(\mathbf{u}) - \sum_{i=1}^n a_{j,i} g_i^-(\mathbf{u})\right\} \\
&\quad - \max_{k\in[m_-]}\left\{\sum_{i=1}^n b_{k,i} g_i^+(\mathbf{u}) - \sum_{i=1}^n b_{k,i} g_i^-(\mathbf{u})\right\} \\
&= \max_{j\in[m_+]}\left\{\sum_{i=1}^n a_{j,i} g_i^+(\mathbf{u}) + \sum_{j'\in[m_+]\setminus\{j\}}\sum_{i=1}^n a_{j,i} g_i^-(\mathbf{u})\right\} - \sum_{j'\in[m_+]}\sum_{i=1}^n a_{j,i} g_i^-(\mathbf{u}) \\
&\quad - \max_{k\in[m_-]}\left\{\sum_{i=1}^n b_{k,i} g_i^+(\mathbf{u}) + \sum_{k'\in[m_-]\setminus\{k\}}\sum_{i=1}^n b_{k,i} g_i^-(\mathbf{u})\right\} + \sum_{k'\in[m_-]}\sum_{i=1}^n b_{k,i} g_i^-(\mathbf{u}) \\
&= \max_{j\in[m_+]}\left\{\sum_{i=1}^n a_{j,i} g_i^+(\mathbf{u}) + \sum_{j'\in[m_+]\setminus\{j\}}\sum_{i=1}^n a_{j,i} g_i^-(\mathbf{u})\right\} + \sum_{k'\in[m_-]}\sum_{i=1}^n b_{k,i} g_i^-(\mathbf{u}) \\
&\quad - \left(\max_{k\in[m_-]}\left\{\sum_{i=1}^n b_{k,i} g_i^+(\mathbf{u}) + \sum_{k'\in[m_-]\setminus\{k\}}\sum_{i=1}^n b_{k,i} g_i^-(\mathbf{u})\right\} + \sum_{j'\in[m_+]}\sum_{i=1}^n a_{j,i} g_i^-(\mathbf{u})\right) \\
&= \max_{j\in[m_+]}\left\{\sum_{i=1}^n a_{j,i} g_i^+(\mathbf{u}) + \sum_{j'\in[m_+]\setminus\{j\}}\sum_{i=1}^n a_{j,i} g_i^-(\mathbf{u}) + \sum_{k'\in[m_-]}\sum_{i=1}^n b_{k,i} g_i^-(\mathbf{u})\right\} \\
&\quad - \max_{k\in[m_-]}\left\{\sum_{i=1}^n b_{k,i} g_i^+(\mathbf{u}) + \sum_{k'\in[m_-]\setminus\{k\}}\sum_{i=1}^n b_{k,i} g_i^-(\mathbf{u}) + \sum_{j'\in[m_+]}\sum_{i=1}^n a_{j,i} g_i^-(\mathbf{u})\right\}
\end{aligned}$$

In each line of the last equality, we recognize a pointwise maximum of a linear combination of pointwise maxima of linear functions. This constitutes a pointwise maximum of linear functions.

This derivation also extends equation (B.2) to the multi-dimensional case by showing an explicit DC decomposition of the output.

B.2 Computing the Feature Vectors

We describe here how to compute the feature vectors in practice. To this end, we show how to construct two (intertwined) neural networks that decompose the objective function into a convex and a concave part. We call these Difference of Convex (DC) networks. Once the DC networks are defined, a standard forward and backward pass in the two networks yields the feature vectors for the convex and concave contribution to the objective function. First, we derive how to perform a DC decomposition in linear and non-linear layers, and then we construct an example of DC networks.

DC Decomposition in a Linear Layer Let W be the weights of a fixed linear layer. We introduce $W^+ = \frac{1}{2}(|W| + W)$ and $W^- = \frac{1}{2}(|W| - W)$. We can note that W^+ and W^- have exclusively non-negative weights, and that $W = W^+ - W^-$. Say we have an input u with the DC decomposition $(u^{\text{cvx}}, u^{\text{ccv}})$, that is: $u = u^{\text{cvx}} - u^{\text{ccv}}$, where both u^{cvx} and u^{ccv} are convex. Then we can decompose the output of the layer as:

$$W \cdot u = \underbrace{(W^+ \cdot u^{\text{cvx}} + W^- \cdot u^{\text{ccv}})}_{\text{convex}} - \underbrace{(W^- \cdot u^{\text{cvx}} + W^+ \cdot u^{\text{ccv}})}_{\text{convex}} \tag{B.1}$$

DC Decomposition in a Piecewise Linear Activation Layer For simplicity purposes, we consider that the non-linear layer is a point-wise maximum across $[K]$ scalar inputs, that is, for an input $(u_k)_{k\in[K]} \in \mathbb{R}^K$, the output is $\max_{k\in[K]} u_k$ (the general multi-dimensional case can be found in Appendix B.1). We suppose that we have a DC decomposition $(u_k^{\text{cvx}}, u_k^{\text{ccv}})$ for each input k. Then we can write the following decomposition for the output of the layer:

$$\begin{aligned}\max_{k\in[K]} u_k &= \max_{k\in[K]} (u_k^{\text{cvx}} - u_k^{\text{ccv}}) \\ &= \underbrace{\max_{k\in[K]} \left(u_k^{\text{cvx}} + \sum_{i\in[K], i\neq k} u_i^{\text{ccv}} \right)}_{\text{convex}} - \underbrace{\sum_{k\in[K]} u_k^{\text{ccv}}}_{\text{convex}}\end{aligned} \tag{B.2}$$

In particular, for a ReLU, we can write:

$$\max(u^{\text{cvx}} - u^{\text{ccv}}, 0) = \underbrace{\max(u^{\text{cvx}}, u^{\text{ccv}})}_{\text{convex}} - \underbrace{u^{\text{ccv}}}_{\text{convex}} \tag{B.3}$$

And for a Max-Pooling layer, one can easily verify that equation (B.2) is equivalent to:

$$MaxPool(u^{\text{cvx}} - u^{\text{ccv}}) = \underbrace{MaxPool(u^{\text{cvx}} - u^{\text{ccv}}) + SumPool(u^{\text{ccv}})}_{\text{convex}} - \underbrace{SumPool(u^{\text{ccv}})}_{\text{convex}} \tag{B.4}$$

An Example of DC Networks We use the previous observations to obtain a DC decomposition in any layer. We now take the example of the neural network used for the experiments on the MNIST data set, and we show how to construct the two neural networks when optimizing W^1, the weights of the first convolutional layer. First let us recall the architecture without decomposition:

Figure B.1: *Detailed network architecture for the MNIST data set.*

We want to optimize the first convolutional layer, therefore we fix all other parameters. Then we apply all operations as described in the previous paragraphs, which yields the DC networks in Figure B.2.

The network graph in Figure B.2 illustrates Proposition 3 for the optimization of W^1: suppose we are interested in $f^{\text{cvx}}(\mathbf{x}, W^1)$, the convex part of the objective function for a given sample $\mathbf{x}$, and we wish to obtain the feature vector needed to perform an update of BCFW. With a forward pass, the oracle for the latent and label variables $(\hat{\mathbf{h}}, \hat{y})$ is efficiently computed; and with a backward pass, we obtain the corresponding feature vector $\Psi(\mathbf{x}, \hat{y}, \hat{\mathbf{h}})$. Indeed, we recall from problem (3.8) that $(\hat{\mathbf{h}}, \hat{y})$ are the latent and label variables maximizing $f^{\text{cvx}}(\mathbf{x}, W^1)$. Then given $\mathbf{x}$, the forward pass in the DC networks sequentially solves the nested maximization: it maximizes the activation of the ReLU and MaxPooling units at each layer,

thereby selecting the best latent variable $\hat{\mathbf{h}}$ at each non-linear layer, and maximizes the output of the SVM layer, thereby selecting the best label $\hat{y}$. At the end of the forward pass, $f^{\text{cvx}}(\mathbf{x}, W^1)$ is therefore available as the output of the convex network, and the feature vector $\Psi(\mathbf{x}, \hat{y}, \hat{\mathbf{h}})$ can be computed as a subgradient of $f^{\text{cvx}}(\mathbf{x}, W^1)$ with respect to W^1.

Linearizing the concave part is equivalent to fixing the activations of the DC networks, which can be done by using a fixed copy of W^1 at the linearization point (all other weights being fixed anyway). Then one can re-use the above reasoning to obtain the feature vectors for the linearized concave part. Altogether, this methodology allows our algorithm to be implemented in any standard deep learning library (our implementation is available at `http://github.com/oval-group/pl-cnn`).

B.3 Experimental Details

Hyper-parameters The hyper-parameters are obtained by cross-validation with a search on powers of 10. In this section, η will denote the initial learning rate. We denote the Softmax + Cross-Entropy loss by SCE, while SVM stands for the usual Support Vector Machines loss.

Table B.1: *Hyper-parameters for the SGD solvers*

	MNIST	CIFAR-10 (SCE)	CIFAR-10 (SVM)	CIFAR-100 (SCE)	CIFAR-100 (SVM)
Adagrad	$\eta = 0.01$ $\lambda = 0.001$	$\eta = 0.01$ $\lambda = 0.001$	$\eta = 0.001$ $\lambda = 0.001$	$\eta = 0.01$ $\lambda = 0.001$	$\eta = 0.001$ $\lambda = 0.001$
Adadelta	$\eta = 1$ $\lambda = 0.001$	$\eta = 1$ $\lambda = 0.001$	$\eta = 0.1$ $\lambda = 0.001$	$\eta = 1$ $\lambda = 0.001$	$\eta = 0.1$ $\lambda = 0.001$
Adam	$\eta = 0.001$ $\lambda = 0.001$	$\eta = 0.001$ $\lambda = 0.001$	$\eta = 0.0001$ $\lambda = 0.001$	$\eta = 0.001$ $\lambda = 0.001$	$\eta = 0.0001$ $\lambda = 0.001$

One may note that the hyper-parameters are the same for both CIFAR-10 and CIFAR-100 for each combination of solver and loss. This makes sense since the initial learning rate mainly depends on the architecture of the network (and not so much on which particular images are fed to this network), which is very similar for the experiments on the CIFAR-10 and CIFAR-100 data sets.

B.4 SVM Formulation & Dual Derivation

Multi-Class SVM Suppose we are given a data set of N samples, for which every sample i has a feature vector $\phi_i \in \mathbb{R}^d$ and a ground truth label $y_i \in \mathcal{Y}$. For every possible label $\bar{y}_i \in \mathcal{Y}$, we introduce the augmented feature vector $\psi_i(\bar{y}_i) \in \mathbb{R}^{|\mathcal{Y}| \times d}$ containing ϕ_i at index $\bar{y}_i$, $-\phi_i$ at index y_i, and zeros everywhere else (then $\psi_i(y_i)$ is just a vector of zeros). We also define $\Delta(\bar{y}_i, y_i)$ as the loss by choosing the output $\bar{y}_i$ instead of the ground truth y_i in our task. For classification, this is the zero-one loss for example.

The SVM optimization problem is formulated as:

$$\begin{aligned} \min_{w,\xi_i} \quad & \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{i=1}^{N}\xi_i \\ \text{subject to:} \quad & \forall i \in [N],\, \forall \bar{y}_i \in \mathcal{Y},\, \xi_i \geq w^T\psi_i(\bar{y}_i) + \Delta(y_i, \bar{y}_i) \end{aligned}$$

Where λ is the regularization hyperparameter. We now add a proximal term to a given starting point w_0:

$$\begin{aligned} \min_{w,\xi_i} \quad & \frac{\lambda}{2}\|w\|^2 + \frac{\mu}{2}\|w - w_0\|^2 + \frac{1}{N}\sum_{i=1}^{N}\xi_i \\ \text{subject to:} \quad & \forall i \in [N],\, \forall \bar{y}_i \in \mathcal{Y},\, \xi_i \geq w^T\psi_i(\bar{y}_i) + \Delta(y_i, \bar{y}_i) \end{aligned}$$

Factorizing the second-order polynomial in w, we obtain the equivalent problem (changed by a constant):

$$\begin{aligned} \min_{w,\xi_i} \quad & \frac{\lambda+\mu}{2}\|w - \frac{\mu}{\lambda+\mu}w_0\|^2 + \frac{1}{N}\sum_{i=1}^{N}\xi_i \\ \text{subject to:} \quad & \forall i \in [N],\, \forall \bar{y}_i \in \mathcal{Y},\, \xi_i \geq w^T\psi_i(\bar{y}_i) + \Delta(y_i, \bar{y}_i) \end{aligned}$$

For simplicity, we introduce the ratio $\rho = \dfrac{\mu}{\lambda+\mu}$.

Dual Objective function The primal problem is:

$$\begin{aligned} \min_{w,\xi_i} \quad & \frac{\lambda+\mu}{2}\|w-\rho w_0\|^2 + \frac{1}{N}\sum_{i=1}^{N}\xi_i \\ \text{subject to:} \quad & \forall i \in [N],\, \forall \bar{y_i} \in \mathcal{Y},\, \xi_i \geq w^T\psi_i(\bar{y_i}) + \Delta(y_i, \bar{y_i}) \end{aligned}$$

The dual problem can be written as:

$$\begin{aligned} \max_{\alpha\geq 0}\min_{w,\xi_i} \quad & \frac{\lambda+\mu}{2}\|w-\rho w_0\|^2 + \frac{1}{N}\sum_{i=1}^{N}\xi_i \\ & + \frac{1}{N}\sum_{i=1}^{N}\sum_{\bar{y_i}\in\mathcal{Y}} \alpha_i(\bar{y_i})\left(\Delta(y_i,\bar{y_i}) + w^T\psi_i(\bar{y_i}) - \xi_i\right) \end{aligned}$$

Then we obtain the following KKT conditions:

$$\begin{aligned} \forall i \in [N],\, \frac{\partial \cdot}{\partial \xi_i} = 0 \longrightarrow & \sum_{\bar{y_i}\in\mathcal{Y}} \alpha_i(\bar{y_i}) = 1 \\ \frac{\partial \cdot}{\partial w} = 0 \longrightarrow & w = \rho w_0 - \underbrace{\frac{1}{N}\frac{1}{\lambda+\mu}\sum_{i=1}^{N}\sum_{\bar{y_i}\in\mathcal{Y}} \alpha_i(\bar{y_i})\psi_i(\bar{y_i})}_{A\alpha} \end{aligned}$$

We also introduce $b = \frac{1}{N}(\Delta(y_i, \bar{y_i}))_{i,\bar{y_i}}$. We define $P_n(\mathcal{Y})$ as the sample-wise probability simplex:

$$\begin{aligned} u \in P_n(\mathcal{Y}) \text{ if:} \quad & \forall i \in [N],\, \forall \bar{y_i} \in \mathcal{Y}, u_i(\bar{y_i}) \geq 0 \\ & \forall i \in [N],\, \sum_{\bar{y_i}\in\mathcal{Y}} u_i(\bar{y_i}) = 1 \end{aligned}$$

We inject back and simplify to:

$$\max_{\alpha\in P_n(\mathcal{Y})} \quad \frac{-(\lambda+\mu)}{2}\|A\alpha\|^2 + \mu w_0^T(A\alpha) + \alpha^T b$$

Finally:

$$\min_{\alpha\in P_n(\mathcal{Y})} \quad f(\alpha)$$

Where:

$$f(\alpha) \triangleq \frac{\lambda+\mu}{2}\|A\alpha\|^2 - \mu w_0^T(A\alpha) - \alpha^T b$$

BCFW derivation We write $\nabla_{(i)} f$ the gradient of f w.r.t. the block (i) of variables in α, padded with zeros on blocks (j) for $j \neq i$. Similarly, $A_{(i)}$ and $b_{(i)}$ contain the rows of A and the elements of b for the block of coordinates (i) and zeros elsewhere. We can write:

$$\nabla_{(i)} f(\alpha) = (\lambda + \mu) A_{(i)}^T A \alpha - \mu A_{(i)} w_0 - b_{(i)}$$

Then the search corner for the block of coordinates (i) is given by:

$$\begin{aligned} s_i &= \underset{s_i'}{\operatorname{argmin}} \Big(< s_i', \nabla_{(i)} f(\alpha) > \Big) \\ &= \underset{s_i'}{\operatorname{argmin}} \Big((\lambda + \mu) \alpha^T A^T A_{(i)} s_i' - \mu w_0^T A_{(i)} s_i' - b_{(i)}^T s_i' \Big) \end{aligned}$$

We replace:

$$\begin{aligned} A\alpha &= \rho w_0 - w \\ A_{(i)} s_i' &= \frac{1}{N} \frac{1}{\lambda + \mu} \sum_{\bar{y}_i \in \mathcal{Y}} s_i'(\bar{y}_i) \psi_i(\bar{y}_i) \\ b_{(i)}^T s_i' &= \frac{1}{N} \sum_{\bar{y}_i \in \mathcal{Y}} s_i'(\bar{y}_i) \Delta(\bar{y}_i, y_i) \end{aligned}$$

We then obtain:

$$\begin{aligned} s_i = \underset{s_i'}{\operatorname{argmin}} \Bigg(& - (w - \rho w_0)^T \sum_{\bar{y}_i \in \mathcal{Y}} s_i'(\bar{y}_i) \psi_i(\bar{y}_i) - w_0^T \rho \sum_{\bar{y}_i \in \mathcal{Y}} s_i'(\bar{y}_i) \psi_i(\bar{y}_i) \\ & - \sum_{\bar{y}_i \in \mathcal{Y}} s_i'(\bar{y}_i) \Delta(\bar{y}_i, y_i) \Bigg) \\ = \underset{s_i'}{\operatorname{argmax}} & \left(w^T \sum_{\bar{y}_i \in \mathcal{Y}} s_i'(\bar{y}_i) \psi_i(\bar{y}_i) + \sum_{\bar{y}_i \in \mathcal{Y}} s_i'(\bar{y}_i) \Delta(\bar{y}_i, y_i) \right) \end{aligned}$$

As expected, this maximum is obtained by setting s_i to one at:

$$y_i^* = \underset{\bar{y}_i \in \mathcal{Y}}{\operatorname{argmax}} \Big(w^T \psi_i(\bar{y}_i) + \Delta(\bar{y}_i, y_i) \Big),$$

and zeros elsewhere. We introduce the notation:

$$\begin{aligned} w_i &= -A_{(i)} \alpha_{(i)} \\ l_i &= b_{(i)}^T \alpha_{(i)} \\ w_s &= -A_{(i)} s_i \end{aligned}$$

$$l_s = b_{(i)}^T s_i$$

Then we have:

$$w_s = -\frac{1}{N}\frac{1}{\lambda+\mu}\psi(y_i^*) = -\frac{1}{N}\frac{1}{\lambda+\mu}\frac{\partial H_i(y_i^*)}{\partial w}$$
$$l_s = \frac{1}{N}\Delta(y_i, y_i^*)$$

The optimal step size in the direction of the block of coordinates (i) is given by :

$$\gamma^* = \underset{\gamma}{\operatorname{argmin}}\, f(\alpha + \gamma(s_i - \alpha_i))$$

The optimal step-size is given by:

$$\gamma^* = \frac{< \nabla_{(i)} f(\alpha), s_i - \alpha_i >}{(\lambda+\mu)\|A(s_i - \alpha_i)\|^2}$$

We introduce $w_d = -A\alpha = w - \rho w_0$. Then we obtain:

$$\begin{aligned}\gamma^* &= \frac{(w_i - w_s)^T(w - \rho w_0) + \rho w_0^T(w_i - w_s) - \frac{1}{\lambda+\mu}(l_i - l_s)}{\|w_i - w_s\|^2} \\ &= \frac{(w_i - w_s)^T w - \frac{1}{\lambda+\mu}(l_i - l_s)}{\|w_i - w_s\|^2}\end{aligned}$$

And the updates are the same as in standard BCFW:

Algorithm 7 *BCFW with warm start*

Let $w^{(0)} = w_0, \quad \forall i \in [N], \quad w_i^{(0)} = 0$
Let $l^{(0)} = 0, \quad \forall i \in [N], \quad l_i^{(0)} = 0$
for k=0...K **do**
  Pick i randomly in $\{1, .., n\}$
  Get $y_i^* = \underset{\bar{y}_i \in \mathcal{Y}}{\operatorname{argmax}} H_i(\bar{y}_i, w^{(k)})$ and $w_s = -\frac{1}{N}\frac{1}{\lambda+\mu}\frac{\partial H_i(y_i^*, w^{(k)})}{\partial w^{(k)}}$
  $l_s = \frac{1}{N}\Delta(y_i^*, y_i)$
  $\gamma = \frac{(w_i - w_s)^T w - \frac{1}{\lambda+\mu}(l_i - l_s)}{\|w_i - w_s\|^2}$ clipped to [0, 1]
  $w_i^{(k+1)} = (1-\gamma)w_i^{(k)} + \gamma w_s$
  $l_i^{(k+1)} = (1-\gamma)l_i^{(k)} + \gamma l_s$
  $w^{(k+1)} = w^{(k)} + w_i^{(k+1)} - w_i^{(k)} = w^{(k)} + \gamma(w_s^{(k)} - w_i^{(k)})$
  $l^{(k+1)} = l^{(k)} + l_i^{(k+1)} - l_i^{(k)}$
end for

In particular, we have proved Proposition (4) in this section: w is initialized to ρw_0 (KKT conditions), and the direction of the conditional gradient, w_s, is given by $\frac{\partial H_i(y_i^*)}{\partial w}$, which is independent of w_0.

Note that the derivation of the Lagrangian dual has introduced a dual variable $\alpha_i(\bar{y}_i)$ for each linear constraint of the SVM problem (this can be replaced by $\alpha_i(\overline{\mathbf{h}}_i, (\bar{y}_i))$ if we consider latent variables). These dual variables indicate the complementary slackness not only for the output class $\bar{y}_i$, but also for each of the activation which defines a piece of the piecewise linear hinge loss. Therefore a choice of α defines a path of activations.

B.5 Sensitivity of SGD Algorithms

Here we discuss some weaknesses of the SGD-based algorithms that we have encountered in practice for our learning objective function. These behaviors have been observed in the case of PL-CNNs, and generally may not appear in different architectures (in particular the failure to learn with high regularization goes away with the use of batch normalization layers).

B.5.1 Initial Learning Rate

As mentioned in the experiments section, the choice of the initial learning rate is critical for good performance of all Adagrad, Adadelta and Adam. When the learning rate is too high, the network does not learn anything and the training and validating accuracies are stuck at random level. When it is too low, the network may take a considerably greater number of epochs to converge.

B.5.2 Failures to Learn

Regularization When the regularization hyper-parameter λ is set to a value of 0.01 or higher on CIFAR-10, SGD solvers get trapped in a local minimum and fail to learn. The SGD solvers indeed fall in the local minimum of shutting down all activations on ReLUs, which provide zero-valued feature vector to the SVM loss layer (and a hinge loss of one). As a consequence, no information can be back-propagated. We plot this behavior below:

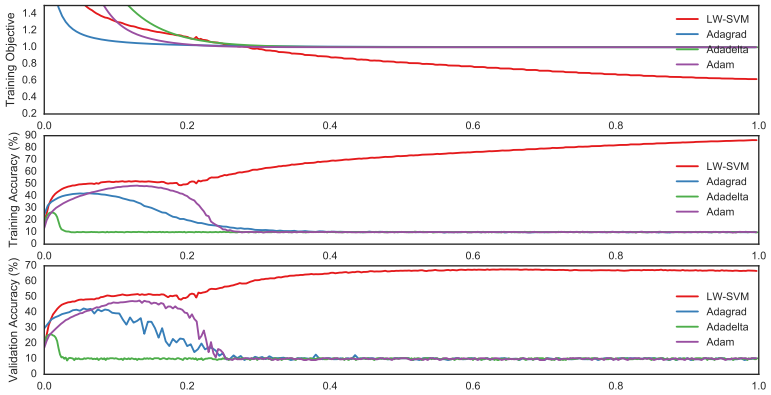


Figure B.3: *Behavior of different algorithms for* $\lambda = 0.01$*. The x-axis has been rescaled to compare the evolution of all algorithms (real training times vary between half an hour to a few hours for the different runs).*

In this situation, the network is at a bad saddle point (note that the training and validation accuracies are stuck at random levels). Our algorithm does not fall into such bad situations, however it is not able to get out of it either: each layer is at a pathological critical point of its own objective function, which makes our algorithm unable to escape from it.

With a lower initial learning rate, the evolution is slower, but eventually the solver goes back to the bad situation presented above.

Biases The same failing behavior as above has been observed when not using the biases in the network. Again our algorithm is robust to this change.

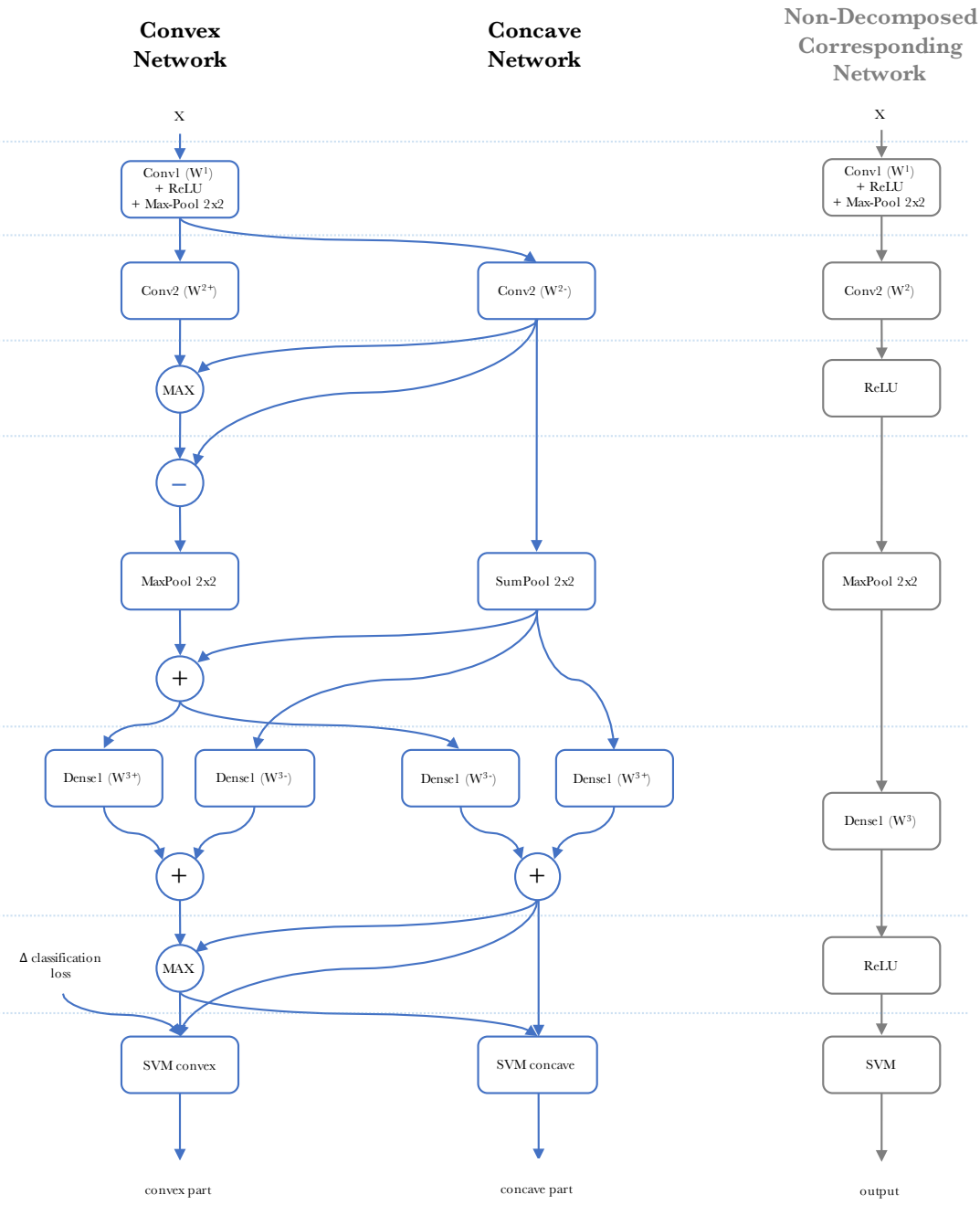


Figure B.2: *Difference of Convex Networks for the optimization of Conv1 in the MNIST architecture. The two leftmost columns represent the DC networks. For each layer, the right column indicates the non-decomposed corresponding operation. Note that we represent the DC decomposition of the SVM layer as unique blocks to keep the graph simple. Given the decomposition method for linear and non-linear layers, one can write down the explicit operations without special difficulty.*

C

Deep Frank-Wolfe For Neural Network Optimization

Contents

C.1 Proofs & Algorithms

For completeness, we prove results for our specific instance of Structural SVM problem. We point out that the proofs of sections C.1.1, C.1.2 and C.1.3 are adaptations from (Lacoste-Julien and Jaggi, 2013). Propositions are numbered according to their appearance in the paper.

C.1.1 Preliminaries

In this section, we assume the loss $\mathcal{L}$ to be a hinge loss:

$$\mathcal{L}_{hinge} : (\boldsymbol{u}, y) \in \mathbb{R}^{|\mathcal{Y}|} \times \mathcal{Y} \mapsto \max \left\{ \max_{\bar{y} \in \mathcal{Y} \setminus \{y\}} \left\{ u_{\bar{y}} + 1 - u_y \right\}, 0 \right\} \tag{C.1}$$

We suppose that we have received a sample $(\boldsymbol{x}, y)$. We simplify the notation $\boldsymbol{f}(\boldsymbol{w}, \boldsymbol{x}) = \boldsymbol{f}_{\boldsymbol{x}}(\boldsymbol{w})$ and $\mathcal{L}(\boldsymbol{u}, y) = \mathcal{L}_y(\boldsymbol{u})$. For simplicity of the notation, and without loss of generality, we consider the proximal problem obtained at time $t = 0$:

$$\min_{\boldsymbol{w} \in \mathbb{R}^p} \left\{ \frac{1}{2\eta} \|\boldsymbol{w} - \boldsymbol{w}_0\|^2 + \mathcal{T}_{\boldsymbol{w}_0} \rho(\boldsymbol{w}) + \mathcal{L}_y \left(\mathcal{T}_{\boldsymbol{w}_0} \boldsymbol{f}_{\boldsymbol{x}}(\boldsymbol{w}) \right) \right\}. \tag{C.2}$$

Let us define the classification task loss:

$$\text{For } (\bar{y}, y) \in \mathcal{Y}^2, \Delta(\bar{y}, y) = \begin{cases} 0 & \text{if } \bar{y} = y, \\ 1 & \text{otherwise.} \end{cases} \tag{C.3}$$

Using this notation, the multi-class hinge loss can be written as:

$$\mathcal{L}_{hinge}(\boldsymbol{u}, y) = \max_{\bar{y} \in \mathcal{Y}} \left\{ u_{\bar{y}} + \Delta(\bar{y}, y) - u_y \right\}. \tag{C.4}$$

Indeed, we can successively write:

$$\begin{aligned} \mathcal{L}_{hinge}(\boldsymbol{u}, y) &= \max \left\{ \max_{\bar{y} \in \mathcal{Y} \setminus \{y\}} \left\{ u_{\bar{y}} + 1 - u_y \right\}, 0 \right\}, \\ &= \max_{\bar{y} \in \mathcal{Y} \setminus \{y\}} \left\{ \max \left\{ u_{\bar{y}} + 1 - u_y, 0 \right\} \right\}, \\ &= \max_{\bar{y} \in \mathcal{Y} \setminus \{y\}} \left\{ \max \left\{ u_{\bar{y}} + \Delta(\bar{y}, y) - u_y, 0 \right\} \right\}, \\ &= \max_{\bar{y} \in \mathcal{Y}} \left\{ \max \left\{ u_{\bar{y}} + \Delta(\bar{y}, y) - u_y, 0 \right\} \right\}, \\ &= \max_{\bar{y} \in \mathcal{Y}} \left\{ u_{\bar{y}} + \Delta(\bar{y}, y) - u_y \right\}. \end{aligned} \tag{C.5}$$

We are now going to re-write problem (C.2) as the sum of a quadratic term and a point-wise maximum of linear functions. For $\bar{y} \in \mathcal{Y}$, let us define:

$$\begin{aligned} \mathbf{a}_{\bar{y}} &= \partial\rho(\boldsymbol{w})\Big|_{\boldsymbol{w}_0} + \partial f_{\boldsymbol{x},\bar{y}}(\boldsymbol{w})\Big|_{\boldsymbol{w}_0} - \partial f_{\boldsymbol{x},y}(\boldsymbol{w})\Big|_{\boldsymbol{w}_0}, \\ b_{\bar{y}} &= \rho(\boldsymbol{w}_0) + f_{\boldsymbol{x},\bar{y}}(\boldsymbol{w}_0) - f_{\boldsymbol{x},y}(\boldsymbol{w}_0) + \Delta(\bar{y}, y). \end{aligned} \tag{C.6}$$

Then we have that:

$$\begin{aligned} &\max_{\bar{y}\in\mathcal{Y}} \left\{ \mathbf{a}_{\bar{y}}^\top (\boldsymbol{w} - \boldsymbol{w}_0) + b_{\bar{y}} \right\} \\ &\quad = \max_{\bar{y}\in\mathcal{Y}} \Big\{ (\partial\rho(\boldsymbol{w})\Big|_{\boldsymbol{w}_0} + \partial f_{\boldsymbol{x},\bar{y}}(\boldsymbol{w})\Big|_{\boldsymbol{w}_0} - \partial f_{\boldsymbol{x},y}(\boldsymbol{w})\Big|_{\boldsymbol{w}_0})^\top (\boldsymbol{w} - \boldsymbol{w}_0) \\ &\qquad + \rho(\boldsymbol{w}_0) + f_{\boldsymbol{x},\bar{y}}(\boldsymbol{w}_0) - f_{\boldsymbol{x},y}(\boldsymbol{w}_0) + \Delta(\bar{y}, y) \Big\}, \\ &\quad = \rho(\boldsymbol{w}_0) + \partial\rho(\boldsymbol{w})\Big|_{\boldsymbol{w}_0}^\top (\boldsymbol{w} - \boldsymbol{w}_0) \\ &\qquad + \max_{\bar{y}\in\mathcal{Y}} \left\{ \partial f_{\boldsymbol{x},\bar{y}}(\boldsymbol{w})\Big|_{\boldsymbol{w}_0}^\top (\boldsymbol{w} - \boldsymbol{w}_0) + f_{\boldsymbol{x},\bar{y}}(\boldsymbol{w}_0) + \Delta(\bar{y}, y) \right\} \\ &\qquad - f_{\boldsymbol{x},y}(\boldsymbol{w}_0) - \partial f_{\boldsymbol{x},y}(\boldsymbol{w})\Big|_{\boldsymbol{w}_0}^\top (\boldsymbol{w} - \boldsymbol{w}_0), \\ &\quad = \mathcal{T}_{\boldsymbol{w}_0}\rho(\boldsymbol{w}) + \mathcal{L}\left(\mathcal{T}_{\boldsymbol{w}_0}\boldsymbol{f}_{\boldsymbol{x}}(\boldsymbol{w}), y\right). \end{aligned} \tag{C.7}$$

Therefore, problem (C.2) can be written as:

$$\min_{\boldsymbol{w}\in\mathbb{R}^p} \left\{ \frac{1}{2\eta}\|\boldsymbol{w} - \boldsymbol{w}_0\|^2 + \max_{\bar{y}\in\mathcal{Y}} \left\{ \mathbf{a}_{\bar{y}}^\top (\boldsymbol{w} - \boldsymbol{w}_0) + b_{\bar{y}} \right\} \right\}. \tag{C.8}$$

We notice that the term $\rho(\boldsymbol{w}_0)$ in $\mathbf{b}$ is a constant that does not depend on $\boldsymbol{w}$ nor $\bar{y}$, therefore we can simplify the expression of $\mathbf{b}$ to:

$$b_{\bar{y}} = f_{\boldsymbol{x},\bar{y}}(\boldsymbol{w}_0) - f_{\boldsymbol{x},y}(\boldsymbol{w}_0) + \Delta(\bar{y}, y). \tag{C.9}$$

We introduce the following notation:

$$\hat{\boldsymbol{w}} = \boldsymbol{w} - \boldsymbol{w}_0, \tag{C.10}$$

$$\mathcal{P} = \{\boldsymbol{\alpha} \in \mathbb{R}_+^{|\mathcal{Y}|} : \sum_{\bar{y}\in\mathcal{Y}} \alpha_{\bar{y}} = 1\}, \tag{C.11}$$

$$A = (\eta \mathbf{a}_{\bar{y}})_{\bar{y}\in\mathcal{Y}} \in \mathbb{R}^{p\times|\mathcal{Y}|}. \tag{C.12}$$

We will also use the indicator vector: $\mathbb{1}_y \in \mathbb{R}^{|\mathcal{Y}|}$, which is equal to 1 at index y and 0 elsewhere.

C.1.2 Dual Objective

Lemma 5 (Dual Objective)**.** *The Lagrangian dual of (C.2) is given by:*

$$\max_{\boldsymbol{\alpha}\in\mathcal{P}} \left\{ -\frac{1}{2\eta}\|A\boldsymbol{\alpha}\|^2 + \mathbf{b}^\top\boldsymbol{\alpha} \right\}. \tag{C.13}$$

Given the dual variables $\boldsymbol{\alpha}$, the primal can be computed as $\hat{\boldsymbol{w}} = -A\boldsymbol{\alpha}$.

Proof. We derive the Lagrangian of the primal problem. For that, we write the problem in the following equivalent ways:

$$\min_{\hat{\boldsymbol{w}}\in\mathbb{R}^p} \left\{ \frac{1}{2\eta}\|\hat{\boldsymbol{w}}\|^2 + \max_{\bar{y}\in\mathcal{Y}} \left\{ \mathbf{a}_{\bar{y}}^\top\hat{\boldsymbol{w}} + b_{\bar{y}} \right\} \right\}, \tag{C.14}$$

$$\min_{\substack{\hat{\boldsymbol{w}}\in\mathbb{R}^p \\ \xi\in\mathbb{R}}} \left\{ \frac{1}{2\eta}\|\hat{\boldsymbol{w}}\|^2 + \xi \right\} \text{ subject to: } \forall\bar{y}\in\mathcal{Y},\ \mathbf{a}_{\bar{y}}^\top\hat{\boldsymbol{w}} + b_{\bar{y}} \leq \xi, \tag{C.15}$$

$$\min_{\substack{\hat{\boldsymbol{w}}\in\mathbb{R}^p \\ \xi\in\mathbb{R}}} \sup_{\boldsymbol{\alpha}\geq 0} \left\{ \frac{1}{2\eta}\|\hat{\boldsymbol{w}}\|^2 + \xi + \sum_{\bar{y}\in\mathcal{Y}} \alpha_{\bar{y}} \left(\mathbf{a}_{\bar{y}}^\top\hat{\boldsymbol{w}} + b_{\bar{y}} - \xi \right) \right\}, \tag{C.16}$$

$$\sup_{\boldsymbol{\alpha}\geq 0} \min_{\substack{\hat{\boldsymbol{w}}\in\mathbb{R}^p \\ \xi\in\mathbb{R}}} \underbrace{\left\{ \frac{1}{2\eta}\|\hat{\boldsymbol{w}}\|^2 + \xi + \sum_{\bar{y}\in\mathcal{Y}} \alpha_{\bar{y}} \left(\mathbf{a}_{\bar{y}}^\top\hat{\boldsymbol{w}} + b_{\bar{y}} - \xi \right) \right\}}_{\Lambda(\hat{\boldsymbol{w}},\xi,\boldsymbol{\alpha})} \quad \text{(by strong duality)}. \tag{C.17}$$

We can now write the KKT conditions of the inner minimization problem:

$$\begin{aligned} \frac{\partial\Lambda(\hat{\boldsymbol{w}},\xi,\boldsymbol{\alpha})}{\partial\xi} = 0: &\quad 1 - \sum_{\bar{y}\in\mathcal{Y}} \alpha_{\bar{y}} = 0, \\ \frac{\partial\Lambda(\hat{\boldsymbol{w}},\xi,\boldsymbol{\alpha})}{\partial\hat{\boldsymbol{w}}} = \mathbf{0}: &\quad \frac{1}{\eta}\hat{\boldsymbol{w}} + \sum_{\bar{y}\in\mathcal{Y}} \alpha_{\bar{y}}\mathbf{a}_{\bar{y}} = \mathbf{0}. \end{aligned} \tag{C.18}$$

This gives $\boldsymbol{\alpha}\in\mathcal{P}$ and $\hat{\boldsymbol{w}} = -A\boldsymbol{\alpha}$, since $A = (\eta\mathbf{a}_{\bar{y}})_{\bar{y}\in\mathcal{Y}}$ by definition. By injecting these constraints in $(C.17)$, we obtain:

$$\max_{\boldsymbol{\alpha}\in\mathcal{P}} \frac{1}{2\eta}\|A\boldsymbol{\alpha}\|^2 + -A\boldsymbol{\alpha}^\top\frac{1}{\eta}A\boldsymbol{\alpha} + \mathbf{b}^\top\boldsymbol{\alpha}, \tag{C.19}$$

which finally gives the desired result. □

C.1.3 Derivation of the Optimal Step-Size

Lemma 6 (Optimal Step-Size)**.** *Suppose that we make a step in the direction of $\boldsymbol{s}\in\mathcal{P}$ in the dual. We define the corresponding primal variables $\boldsymbol{w}_{\boldsymbol{s}} = -A\boldsymbol{s}$ and $\lambda_{\boldsymbol{s}} = \mathbf{b}^\top\boldsymbol{s}$, as well as $\lambda = \mathbf{b}^\top\boldsymbol{\alpha}$. Then the optimal step-size is given by:*

$$\gamma = \frac{(\boldsymbol{w} - \boldsymbol{w}_0 - \boldsymbol{w}_{\boldsymbol{s}})^\top(\boldsymbol{w} - \boldsymbol{w}_0) + \eta(\lambda_{\boldsymbol{s}} - \lambda)}{\|\boldsymbol{w} - \boldsymbol{w}_0 - \boldsymbol{w}_{\boldsymbol{s}}\|^2}. \tag{C.20}$$

Proof. Given the direction $\boldsymbol{s}$, we take the step $\boldsymbol{\alpha} + \gamma(\boldsymbol{s} - \boldsymbol{\alpha})$. The new objective is given by:

$$-\frac{1}{2\eta}\|A(\boldsymbol{\alpha} + \gamma(\boldsymbol{s} - \boldsymbol{\alpha}))\|^2 + \mathbf{b}^\top(\boldsymbol{\alpha} + \gamma(\boldsymbol{s} - \boldsymbol{\alpha})). \tag{C.21}$$

In order to compute the optimal step-size, we compute the derivative of the above expression with respect to gamma, and set it to 0:

$$-\frac{1}{\eta}(\boldsymbol{s} - \boldsymbol{\alpha})^\top A^\top A(\boldsymbol{\alpha} + \gamma(\boldsymbol{s} - \boldsymbol{\alpha})) + \mathbf{b}^\top(\boldsymbol{s} - \boldsymbol{\alpha}) = 0. \tag{C.22}$$

We can isolate the unique term containing γ:

$$-\frac{1}{\eta}\gamma\|A(\boldsymbol{s} - \boldsymbol{\alpha})\|^2 - \frac{1}{\eta}(\boldsymbol{s} - \boldsymbol{\alpha})^\top A^\top A\boldsymbol{\alpha} + \mathbf{b}^\top(\boldsymbol{s} - \boldsymbol{\alpha}) = 0. \tag{C.23}$$

This yields:

$$\begin{aligned}
\gamma &= \frac{-\frac{1}{\eta}(\boldsymbol{s} - \boldsymbol{\alpha})^\top A^\top A\boldsymbol{\alpha} + \mathbf{b}^\top(\boldsymbol{s} - \boldsymbol{\alpha})}{\frac{1}{\eta}\|A(\boldsymbol{s} - \boldsymbol{\alpha})\|^2}, \\
&= \frac{-\frac{1}{\eta}(A\boldsymbol{s} - A\boldsymbol{\alpha})^\top A\boldsymbol{\alpha} + \mathbf{b}^\top(\boldsymbol{s} - \boldsymbol{\alpha})}{\frac{1}{\eta}\|A\boldsymbol{s} - A\boldsymbol{\alpha}\|^2}, \\
&= \frac{-(A\boldsymbol{s} - A\boldsymbol{\alpha})^\top A\boldsymbol{\alpha} + \eta\mathbf{b}^\top(\boldsymbol{s} - \boldsymbol{\alpha})}{\|A\boldsymbol{s} - A\boldsymbol{\alpha}\|^2}.
\end{aligned} \tag{C.24}$$

We can then inject the primal variables and simplify:

$$\begin{aligned}
\gamma &= \frac{(-\boldsymbol{w}_s + \hat{\boldsymbol{w}})^\top\hat{\boldsymbol{w}} + \eta(\lambda_s - \lambda)}{\|-\boldsymbol{w}_s + \hat{\boldsymbol{w}}\|^2}, \\
&= \frac{(\boldsymbol{w} - \boldsymbol{w}_0 - \boldsymbol{w}_s)^\top(\boldsymbol{w} - \boldsymbol{w}_0) + \eta(\lambda_s - \lambda)}{\|\boldsymbol{w} - \boldsymbol{w}_0 - \boldsymbol{w}_s\|^2}.
\end{aligned} \tag{C.25}$$

□

C.1.4 Primal-Dual Proximal Frank-Wolfe Algorithm

We present here the primal-dual algorithm that solves $(C.2)$ using the previous results:

Algorithm 8 *Proximal Frank Wolfe Algorithm*

Require: proximal coefficient η, initial point $\boldsymbol{w}_0 \in \mathbb{R}^p$, sample $(\boldsymbol{x}, y)$.

1: $\boldsymbol{w}_1 = \boldsymbol{w}_0 - \eta \partial \rho(\boldsymbol{w})\Big|_{\boldsymbol{w}_0}$ ▷ Initialization $\boldsymbol{w}_0 - A\boldsymbol{\alpha}$ with $\boldsymbol{\alpha} = \mathbf{1}_y$

2: $\lambda_1 = 0$ ▷ Initialization $\mathbf{b}^\top \boldsymbol{\alpha}$ with $\boldsymbol{\alpha} = \mathbf{1}_y$

3: $t = 1$

4: **while** not converged **do**

5: Choose direction $\boldsymbol{s}_t \in \mathcal{P}$ ▷ (e.g. conditional gradient or smoothed loss)

6: $\boldsymbol{w}_{\boldsymbol{s}} = -A\boldsymbol{s}_t$

7: $\lambda_{\boldsymbol{s}} = \mathbf{b}^\top \boldsymbol{s}_t$

8: $\gamma_t = \dfrac{(\boldsymbol{w}_t - \boldsymbol{w}_0 - \boldsymbol{w}_{\boldsymbol{s}})^\top (\boldsymbol{w}_t - \boldsymbol{w}_0) + \eta(\lambda_{\boldsymbol{s}} - \lambda_t)}{\|\boldsymbol{w} - \boldsymbol{w}_0 - \boldsymbol{w}_{\boldsymbol{s}}\|^2}$ ▷ Optimal- step-size

9: $\boldsymbol{w}_{t+1} = (1 - \gamma_t)\boldsymbol{w}_t + \gamma_t(\boldsymbol{w}_{\boldsymbol{s}} + \boldsymbol{w}_0)$ ▷ $A\boldsymbol{\alpha}_{t+1} = (1 - \gamma_t)A\boldsymbol{\alpha}_t + \gamma_t A\boldsymbol{s}_t$

10: $\lambda_{t+1} = (1 - \gamma_t)\lambda_t + \gamma_t \lambda_{\boldsymbol{s}}$ ▷ $\mathbf{b}^\top \boldsymbol{\alpha}_{t+1} = (1 - \gamma_t)\mathbf{b}^\top \boldsymbol{\alpha}_t + \gamma_t \mathbf{b}^\top \boldsymbol{s}_t$

11: $t = t + 1$

12: **end while**

Note that when $\boldsymbol{f}_{\boldsymbol{x}}$ is linear, and when the search direction $\boldsymbol{s}$ is given by the conditional gradient, we recover the standard Frank-Wolfe algorithm for SVM (Lacoste-Julien and Jaggi, 2013).

C.1.5 Single-Step Proximal Frank-Wolfe Algorithm

We now provide some simplification to the steps 6, 8 and 9 of Algorithm 8 when a single step is taken, as is the case in the DFW algorithm. This corresponds to the iteration $t = 1$.

Theorem 2 (Cost per iteration)**.** *If a single step is performed on the dual of (4.6), its conditional gradient is given by* $-\partial \left(\rho(\boldsymbol{w}) + \mathcal{L}_y(\boldsymbol{f}_{\boldsymbol{x}}(\boldsymbol{w}))\right)\Big|_{\boldsymbol{w}_t}$*. The resulting update can be written as:*

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \left[\partial \rho(\boldsymbol{w})\Big|_{\boldsymbol{w}_t} + \gamma \partial \mathcal{L}_j(\boldsymbol{f}_j(\boldsymbol{w}))\Big|_{\boldsymbol{w}_t}\right] \tag{C.26}$$

Proof. It is known that for linear SVMs, the direction of the dual conditional gradient is given by the negative sub-gradient of the primal (Lacoste-Julien and Jaggi, 2013; Bach, 2015). We apply this result to the Taylor expansion of the network, which is the local model used for the proximal problem. Then we have that at iteration $t = 1$, the conditional gradient is given by:

$$-\partial \left(\mathcal{T}_{\boldsymbol{w}_0} \rho(\boldsymbol{w}) + \mathcal{L}_y(\mathcal{T}_{\boldsymbol{w}_0} \boldsymbol{f}_{\boldsymbol{x}}(\boldsymbol{w}))\right)\Big|_{\boldsymbol{w}_0}. \tag{C.27}$$

It now suffices to notice that a first-order Taylor expansion does not modify the derivative at its point of linearization: for a function ϕ, $\partial T_{\boldsymbol{w}_0}\phi(\boldsymbol{w})\Big|_{\boldsymbol{w}_0} = \partial\phi(\boldsymbol{w})\Big|_{\boldsymbol{w}_0}$. By applying this property and the chain rule to (C.27), we obtain that the conditional gradient is given by:

$$-\partial\left(\rho(\boldsymbol{w}) + \mathcal{L}_y(\boldsymbol{f}_{\boldsymbol{x}}(\boldsymbol{w}))\right)\Big|_{\boldsymbol{w}_0}. \tag{C.28}$$

This completes the proof that the conditional gradient direction is given by a stochastic gradient. We now prove equation (C.26) in the next lemma. □

Lemma 7. *Suppose that we apply the Proximal Frank-Wolfe algorithm with a single step. Let* $\boldsymbol{\delta}_t = \partial\left[\boldsymbol{s}_t^\top(f_{\boldsymbol{x},\bar{y}}(\boldsymbol{w}_0) - f_{\boldsymbol{x},y}(\boldsymbol{w}_0))_{\bar{y}\in\mathcal{Y}}\right]$ *and* $\mathrm{r}_t = \partial_w\rho(\boldsymbol{w}_0)$. *Then we can rewrite step 6 as:*

$$\boldsymbol{w}_s = -\eta\left[\mathrm{r}_t + \boldsymbol{\delta}_t\right]. \tag{C.29}$$

In addition, we can simplify steps 8 and 9 of Algorithm 8 to:

$$\gamma_t = \frac{-\eta\boldsymbol{\delta}_t^\top\mathrm{r}_t + \boldsymbol{s}_t^\top\mathbf{b}}{\eta\|\boldsymbol{\delta}_t\|^2} \text{ clipped to [0, 1]}, \tag{C.30}$$

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_0 - \eta\left[\mathrm{r}_t + \gamma_t\boldsymbol{\delta}_t\right]. \tag{C.31}$$

Proof. Again, since we perform a single step of FW, we assume $t = 1$. To prove equation (C.29), we note that:

$$\begin{aligned}\boldsymbol{w}_s &= -A\boldsymbol{s},\\ &= -\eta\left[\partial\rho(\boldsymbol{w})\Big|_{\boldsymbol{w}_0} + \left((\partial f_{\boldsymbol{x},\bar{y}}(\boldsymbol{w})\Big|_{\boldsymbol{w}_0} - \partial f_{\boldsymbol{x},y}(\boldsymbol{w})\Big|_{\boldsymbol{w}_0})_{\bar{y}\in\mathcal{Y}}\right)^\top\boldsymbol{s}_t\right],\\ &= -\eta\left[\mathrm{r}_t + \boldsymbol{\delta}_t\right].\end{aligned} \tag{C.32}$$

We point out the two following results:

$$\boldsymbol{w}_t - \boldsymbol{w}_0 = \boldsymbol{w}_1 - \boldsymbol{w}_0 = -\eta\partial\rho(\boldsymbol{w})\Big|_{\boldsymbol{w}_0} = -\eta\mathrm{r}_t, \tag{C.33}$$

and:

$$\boldsymbol{w}_t - \boldsymbol{w}_0 - \boldsymbol{w}_s = -\eta\mathrm{r}_t + \eta\mathrm{r}_t + \eta\boldsymbol{\delta}_t = \eta\boldsymbol{\delta}_t. \tag{C.34}$$

Since $\lambda_1 = 0$ by definition, equation (C.30) is obtained with a simple application of equations C.33 and C.34. Finally, we prove equation C.31 by writing:

$$\begin{aligned} \boldsymbol{w}_{t+1} &= (1-\gamma_t)\boldsymbol{w}_t + \gamma_t(\boldsymbol{w}_{\boldsymbol{s}} + \boldsymbol{w}_0), \\ &= (1-\gamma_t)(\boldsymbol{w}_0 - \eta \mathrm{r}_t) + \gamma_t(-\eta \mathrm{r}_t - \eta\boldsymbol{\delta}_t + \boldsymbol{w}_0), \\ &= \boldsymbol{w}_0 - \eta(\mathrm{r}_t + \gamma_t\boldsymbol{\delta}_t). \end{aligned} \tag{C.35}$$

□

C.1.6 Smoothing the Loss

As pointed out in the paper, the SVM loss is non-smooth and has sparse derivatives, which can prevent the effective training of deep neural networks (Berrada et al., 2018). Partial linearization can solve this problem by locally smoothing the dual (Mohapatra et al., 2016). However, this would introduce a temperature hyper-parameter which is undesirable. Therefore, we note that DFW can be applied with any direction that is feasible in the dual, since it computes an optimal step-size. In particular, the following result states that we can use the well-conditioned and non-sparse gradient of cross-entropy.

Proposition 17. *The gradient of cross-entropy in the primal gives a feasible direction in the dual. Furthermore, we can inexpensively detect when this feasible direction cannot provide any improvement in the dual, and automatically switch to the conditional gradient when that is the case.*

For simplicity, we divide Proposition 17 into two distinct parts: first we show how the CE gradient gives a feasible direction in the dual, and then how it can be detected to be an ascent direction.

Lemma 8. *The gradient of cross-entropy in the primal gives a feasible direction in the dual. In other words, the gradient of cross-entropy $\boldsymbol{g}$ in the primal is such that there exists a dual search direction $\boldsymbol{s} \in \mathcal{P}$ verifying $\boldsymbol{g} = -A\boldsymbol{s}$.*

Proof. We consider the vector of scores $(f_{\boldsymbol{x},\bar{y}}(\boldsymbol{w}))_{\bar{y}\in\mathcal{Y}} \in \mathbb{R}^{|\mathcal{Y}|}$. We compute its softmax: $\boldsymbol{s}_{\text{ce}} = \left(\frac{\exp(f_{\boldsymbol{x},\bar{y}}(\boldsymbol{w}))}{\sum_{j\in\mathcal{Y}}\exp(f_{\boldsymbol{x},j}(\boldsymbol{w}))}\right)_{\bar{y}\in\mathcal{Y}}$. Clearly, $\boldsymbol{s}_{\text{ce}} \in \mathcal{P}$ by property of the softmax.

Furthermore, by going back to the definition of A, one can easily verify that $-A\boldsymbol{s}_{\text{ce}}$ is exactly the primal gradient given by a backward pass through the cross-entropy loss instead of the hinge loss. This concludes the proof. □

The previous lemma has shown that we can use the gradient of cross-entropy as a feasible direction $\boldsymbol{s}_{\text{ce}}$ in the dual. The next step is to make it a dual ascent direction, that is a direction which always permits improvement on the dual objective (unless at the optimal point). In what follows, we show that we can inexpensively (approximately) compute a sufficient condition for $\boldsymbol{s}_{\text{ce}}$ to be an ascent direction. If the condition is not satisfied, then we can automatically switch to use the subgradient of the hinge loss (which is known as an ascent direction in the dual).

Lemma 9. *Let $\boldsymbol{s} \in \mathcal{P}$ be a feasible direction in the dual, and also let $\boldsymbol{v} = (\mathcal{T}_{\boldsymbol{w}_0}\boldsymbol{f}_{\boldsymbol{x}}(\boldsymbol{w}_t)_{\bar{y}} + \Delta(\bar{y}, y) - \mathcal{T}_{\boldsymbol{w}_0}\boldsymbol{f}_{\boldsymbol{x}}(\boldsymbol{w}_t)_y)_{\bar{y}\in\mathcal{Y}} \in \mathbb{R}^{|\mathcal{Y}|}$ be the vector of augmented scores output by the linearized model. Let us assume that we apply the single-step Proximal Frank-Wolfe algorithm (that is, we have $t = 1$), and that ρ is a non-negative function.*

Then $\boldsymbol{s}^\top\boldsymbol{v} > 0$ is a sufficient condition for $\boldsymbol{s}$ to be an ascent direction in the dual.

Proof. Let $\boldsymbol{s} \in \mathcal{P}$, $\boldsymbol{v} = (\mathcal{T}_{\boldsymbol{w}_0}\boldsymbol{f}_{\boldsymbol{x}}(\boldsymbol{w}_t)_{\bar{y}} + \Delta(\bar{y}, y) - \mathcal{T}_{\boldsymbol{w}_0}\boldsymbol{f}_{\boldsymbol{x}}(\boldsymbol{w}_t)_y)_{\bar{y}\in\mathcal{Y}}$. By definition, we have that:

$$\begin{aligned}\boldsymbol{v} &= \left(\mathbf{a}_{\bar{y}}^\top(\boldsymbol{w}_t - \boldsymbol{w}_0) + b_{\bar{y}} - \mathcal{T}_{\boldsymbol{w}_0}\rho(\boldsymbol{w})\right)_{\bar{y}\in\mathcal{Y}}, \\ &= \frac{1}{\eta}A^\top(\boldsymbol{w}_t - \boldsymbol{w}_0) + \mathbf{b} - (\mathcal{T}_{\boldsymbol{w}_0}\rho(\boldsymbol{w}))_{\bar{y}\in\mathcal{Y}}.\end{aligned} \tag{C.36}$$

Therefore:

$$
\begin{aligned}
& \boldsymbol{s}^\top \boldsymbol{v} > 0 \\
\iff & \frac{1}{\eta}(A\boldsymbol{s})^\top(\boldsymbol{w}_t - \boldsymbol{w}_0) + \boldsymbol{s}^\top \mathbf{b} - \boldsymbol{s}^\top \left(\mathcal{T}_{\boldsymbol{w}_0}\rho(\boldsymbol{w})\right)_{\bar{y}\in\mathcal{Y}} > 0, \\
\iff & (A\boldsymbol{s})^\top(\boldsymbol{w}_t - \boldsymbol{w}_0) + \eta \boldsymbol{s}^\top \mathbf{b} - \eta \mathcal{T}_{\boldsymbol{w}_0}\rho(\boldsymbol{w}) > 0, \quad \text{(since } \boldsymbol{s} \in \mathcal{P} \text{ and } \eta > 0) \\
\iff & -\boldsymbol{w}_s^\top(\boldsymbol{w}_t - \boldsymbol{w}_0) + \eta \boldsymbol{s}^\top \mathbf{b} - \eta\rho(\boldsymbol{w}_0) - \eta \partial\rho(\boldsymbol{w}_0)^\top(\boldsymbol{w}_t - \boldsymbol{w}_0) > 0, \\
\iff & -\boldsymbol{w}_s^\top(\boldsymbol{w}_t - \boldsymbol{w}_0) + \eta \boldsymbol{s}^\top \mathbf{b} - \eta\rho(\boldsymbol{w}_0) + (\boldsymbol{w}_t - \boldsymbol{w}_0)^\top(\boldsymbol{w}_t - \boldsymbol{w}_0) > 0, \\
\iff & (\boldsymbol{w}_t - \boldsymbol{w}_0 - \boldsymbol{w}_s)^\top(\boldsymbol{w}_t - \boldsymbol{w}_0) + \eta \boldsymbol{s}^\top \mathbf{b} - \eta\rho(\boldsymbol{w}_0) > 0, \\
\implies & (\boldsymbol{w}_t - \boldsymbol{w}_0 - \boldsymbol{w}_s)^\top(\boldsymbol{w}_t - \boldsymbol{w}_0) + \eta \boldsymbol{s}^\top \mathbf{b} > 0, \quad \text{(because } \rho(\boldsymbol{w}_0) \geq 0) \\
\iff & \gamma_t > 0 \quad \text{(we have that } \lambda_t = 0 \text{ at } t = 1).
\end{aligned}
\tag{C.37}
$$

We have just shown that if $\boldsymbol{s}^\top \boldsymbol{v} > 0$, then $\gamma_t > 0$. Since γ_t is an optimal step-size, this indicates that $\boldsymbol{s}$ is an ascent direction (we would obtain $\gamma_t = 0$ for a direction $\boldsymbol{s}$ that cannot provide improvement). □

Approximate Condition. In practice, we consider that $\mathcal{T}_{\boldsymbol{w}_0}\boldsymbol{f}_{\boldsymbol{x}}(\boldsymbol{w}_t) \simeq \boldsymbol{f}_{\boldsymbol{x}}(\boldsymbol{w}_0)$. Indeed, for $t = 1$, we have that $\|\mathcal{T}_{\boldsymbol{w}_0}\boldsymbol{f}_{\boldsymbol{x}}(\boldsymbol{w}) - \boldsymbol{f}_{\boldsymbol{x}}(\boldsymbol{w}_0)\| = \mathcal{O}(\|\boldsymbol{w}_t - \boldsymbol{w}_0\|)$, and $\|\boldsymbol{w}_t - \boldsymbol{w}_0\| = \|\eta\partial_w\rho(\boldsymbol{w}_0))\|$, which is typically very small (we use a weight decay coefficient in the order of 1E^{-4} in our experimental settings). Therefore, we replace $\mathcal{T}_{\boldsymbol{w}_0}\boldsymbol{f}_{\boldsymbol{x}}(\boldsymbol{w})$ by $\boldsymbol{f}_{\boldsymbol{x}}(\boldsymbol{w}_0)$ in the above criterion, which becomes inexpensive since $\boldsymbol{f}_{\boldsymbol{x}}(\boldsymbol{w}_0)$ is already computed by the forward pass.

C.1.7 Nesterov Momentum

As can be seen in the previous primal-dual algorithms, taking a step in the dual can be decomposed into two stages: the initialization and the movement along the search direction. The initialization step is not informative about the optimization problem. Therefore, we discard it from the momentum velocity, and only accumulate the step along the conditional gradient (scaled by $\gamma_t\eta$). This results in the following velocity update:

$$
\mathbf{z}_{t+1} = \mu\mathbf{z}_t - \eta\gamma_t(\mathrm{r}_t + \boldsymbol{\delta}_t). \tag{C.38}
$$

C.2 Experimental Details on the CIFAR Data Sets

C.2.1 Adaptive Gradient Baselines: Cross-Validation (Without Data Augmentation)

l_2	η	CIFAR-10 Accuracy (%)	CIFAR-100 Accuracy (%)
0.0001	0.001	71.6	39.44
0.0001	**0.01**	**88.18**	**55.72**
0.0001	0.1	86.4	55.44
0.0001	1	68.48	20.68

Table C.1: Cross-Validation for ADAGRAD on DN architecture (best validation accuracy obtained during training).

l_2	η	CIFAR-10 Accuracy (%)	CIFAR-100 Accuracy (%)
0.0001	0.001	68.98	31.86
0.0001	**0.01**	**86.4**	53.82
0.0001	0.1	83.6	51.18
0.0005	0.001	68.66	32.5
0.0005	**0.01**	86.3	**56.16**
0.0005	0.1	77.92	44.12

Table C.2: Cross-Validation for ADAGRAD on WRN architecture (best validation accuracy obtained during training).

l_2	η	CIFAR-10 Accuracy (%)	CIFAR-100 Accuracy (%)
0.0001	0.0001	86.26	50.7
0.0001	**0.001**	**89.42**	**63.9**
0.0001	0.01	81.12	51.82

Table C.3: Cross-Validation for ADAM on DN architecture (best validation accuracy obtained during training).

l_2	η	CIFAR-10 Accuracy (%)	CIFAR-100 Accuracy (%)
0.0001	0.0001	79.7	41.42
0.0001	**0.001**	**86.1**	**58.7**
0.0001	0.01	80.06	50.86
0.0005	0.0001	78.88	40.08
0.0005	0.001	85.14	55.26
0.0005	0.01	72.54	36.82

Table C.4: Cross-Validation for ADAM on WRN architecture (best validation accuracy obtained during training).

l_2	η	CIFAR-10 Accuracy (%)	CIFAR-100 Accuracy (%)
0.0001	0.0001	84.28	49.54
0.0001	**0.001**	**90.4**	**68.54**
0.0001	0.01	83.98	50.44

Table C.5: Cross-Validation for AMSGRAD on DN architecture (best validation accuracy obtained during training).

l_2	η	CIFAR-10 Accuracy (%)	CIFAR-100 Accuracy (%)
0.0001	0.0001	75.86	41.6
0.0001	**0.001**	**87.02**	**59.6**
0.0001	0.01	82.32	52.12
0.0005	0.0001	75.74	42.28
0.0005	0.001	86.16	57.82
0.0005	0.01	75.82	36.48

Table C.6: Cross-Validation for AMSGRAD on WRN architecture (best validation accuracy obtained during training).

l_2	η	CIFAR-10 Accuracy (%)	CIFAR-100 Accuracy (%)
0.0001	0.001	72.72	40.96
0.0001	0.01	83.26	53.12
0.0001	**0.1**	**91.7**	**59.7**
0.0001	1	10.16	1.16

Table C.7: Cross-Validation for BPGRAD on DN architecture (best validation accuracy obtained during training).

l_2	η	CIFAR-10 Accuracy (%)	CIFAR-100 Accuracy (%)
0.0001	0.001	64.98	31.9
0.0001	0.01	78.46	44.26
0.0001	**0.1**	**89.24**	54.42
0.0001	1	16.1	1.16
0.0005	0.001	68.08	33.26
0.0005	**0.01**	85.44	**59.9**
0.0005	0.1	88.44	51.28
0.0005	1	10.16	1.16

Table C.8: Cross-Validation for BPGRAD on WRN architecture (best validation accuracy obtained during training).

C.2.2 Adaptive Gradient Baselines: Cross-Validation (With Data Augmentation)

l_2	η	batchsize	CIFAR-10 Accuracy (%)	CIFAR-100 Accuracy (%)
0.0001	0.0001	64	90.38	61.6
0.0001	0.0001	128	87.86	57.82
0.0001	0.0001	256	86.66	53.64
0.0001	**0.001**	**64**	92.52	**69.66**
0.0001	**0.001**	**128**	**92.72**	69.5
0.0001	0.001	256	92.64	67.56
0.0001	0.01	64	82.1	45
0.0001	0.01	128	83.9	53.4
0.0001	0.01	256	86.86	58.1

Table C.9: Cross-Validation for AMSGRAD on DN architecture with data augmentation (best validation accuracy obtained during training).

l_2	η	batchsize	CIFAR-10 Accuracy (%)	CIFAR-100 Accuracy (%)
0.0001	0.0001	128	90.5	64.36
0.0001	0.0001	256	89.6	62.02
0.0001	0.0001	512	88.26	58.68
0.0001	**0.001**	**128**	91.7	**69.3**
0.0001	0.001	256	91.8	68.98
0.0001	**0.001**	**512**	**91.88**	68.64
0.0001	0.01	128	83.36	53.72
0.0001	0.01	256	84.58	57.28
0.0001	0.01	512	87.42	60.68
0.0005	0.0001	128	91.44	65.52
0.0005	0.0001	256	89.7	61.98
0.0005	0.0001	512	88.48	59.1
0.0005	0.001	128	90.82	67.38
0.0005	0.001	256	91	67.58
0.0005	0.001	512	91.06	67.06
0.0005	0.01	128	72.6	34.8
0.0005	0.01	256	76.56	41.82
0.0005	0.01	512	79.12	45.6

Table C.10: Cross-Validation for AMSGRAD on WRN architecture with data augmentation (best validation accuracy obtained during training).

l_2	η	batchsize	CIFAR-10 Accuracy (%)	CIFAR-100 Accuracy (%)
0.0001	0.01	64	92.8	69.12
0.0001	0.01	128	91.38	66.26
0.0001	0.01	256	89.46	60.68
0.0001	**0.1**	**64**	**95.34**	68.04
0.0001	0.1	64	95.34	66.78
0.0001	0.1	128	94.7	73.62
0.0001	**0.1**	**128**	94.7	**74.1**
0.0001	0.1	256	94	70.9
0.0001	1	64	77.04	38.44
0.0001	1	128	82.56	52.12
0.0001	1	256	87.38	60.6
0.0001	10	64	10.16	1.16
0.0001	10	128	10.16	1.16
0.0001	10	256	10.56	1.62

Table C.11: Cross-Validation for DFW on DN architecture with data augmentation (best validation accuracy obtained during training).

l_2	η	batchsize	CIFAR-10 Accuracy (%)	CIFAR-100 Accuracy (%)
0.0001	0.01	128	93.24	71.18
0.0001	0.01	256	91.8	67.36
0.0001	0.01	512	90.9	64.74
0.0001	0.1	128	94.18	74.26
0.0001	0.1	256	94.66	73.24
0.0001	0.1	512	94.02	71.66
0.0001	1	128	84.7	55.1
0.0001	1	256	89.62	61.88
0.0001	**1**	**512**	**94.98**	73.94
0.0001	10	128	10.72	1.32
0.0001	10	256	10.72	4.96
0.0001	10	512	13.62	6.4
0.0005	0.01	128	94.1	72.14
0.0005	0.01	256	92.72	69.96
0.0005	0.01	512	90.84	64.04
0.0005	0.1	128	88.86	63.06
0.0005	**0.1**	**256**	94.68	**75.34**
0.0005	0.1	512	94.1	72.54
0.0005	1	128	63.3	26.9
0.0005	1	256	72.08	38.28
0.0005	1	512	80.74	48.1
0.0005	1	512	80.74	44.52
0.0005	10	128	10.72	1.36
0.0005	10	256	10.72	1.3
0.0005	10	512	14.18	1.98

Table C.12: Cross-Validation for DFW on WRN architecture with data augmentation (best validation accuracy obtained during training).

C.2.3 Convergence Plots

In this section we provide the convergence plots of the different algorithms on the CIFAR data sets without data augmentation. In some cases the training performance can show some oscillations. We emphasize that this is the result of cross-validating the initial learning rate based on the validation set performance: sometimes a better-behaved convergence would be obtained on the training set with a lower learning rate. However this lower learning rate is not selected because it does not provide the best validation performance.

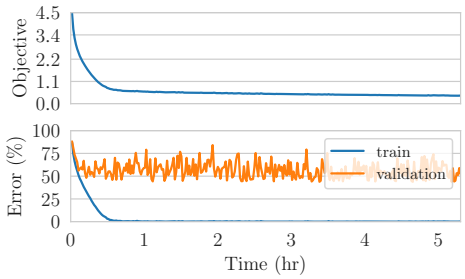


Figure C.1: *Convergence plot of Adagrad on CIFAR 100 with DN architecture.*

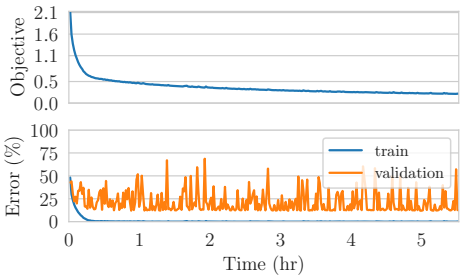


Figure C.2: *Convergence plot of Adagrad on CIFAR 10 with DN architecture.*

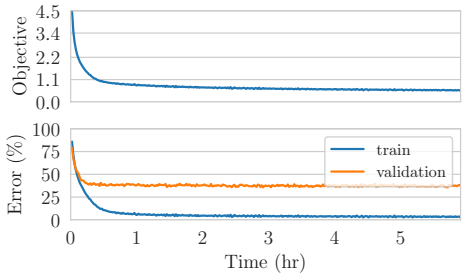


Figure C.3: *Convergence plot of Adam on CIFAR 100 with DN architecture.*

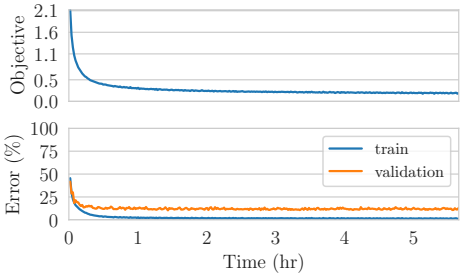


Figure C.4: *Convergence plot of Adam on CIFAR 10 with DN architecture.*

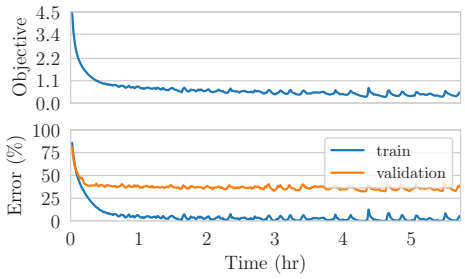


Figure C.5: *Convergence plot of AMSGrad on CIFAR 100 with DN architecture.*

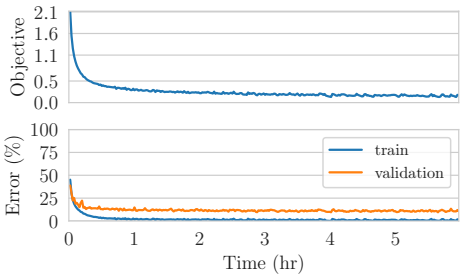


Figure C.6: *Convergence plot of AMSGrad on CIFAR 10 with DN architecture.*

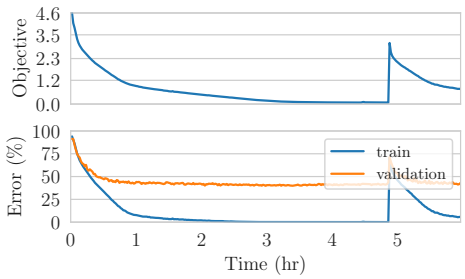


Figure C.7: *Convergence plot of BPGrad on CIFAR 100 with DN architecture.*

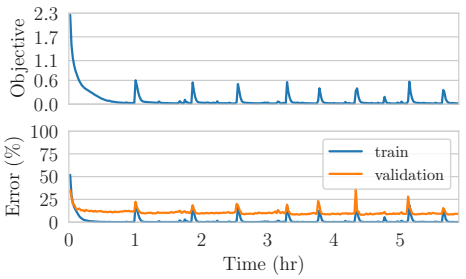


Figure C.8: *Convergence plot of BPGrad on CIFAR 10 with DN architecture.*

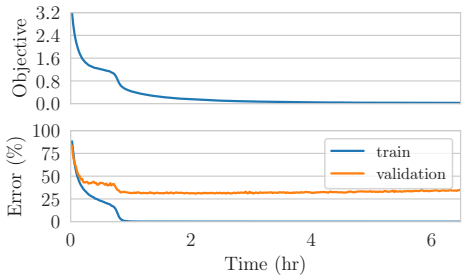


Figure C.9: *Convergence plot of DFW on CIFAR 100 with DN architecture.*

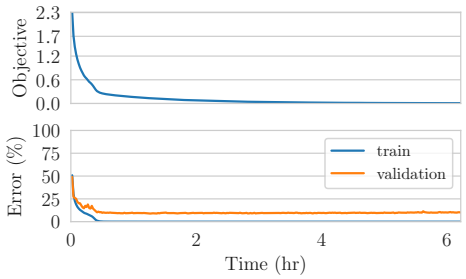


Figure C.10: *Convergence plot of DFW on CIFAR 10 with DN architecture.*

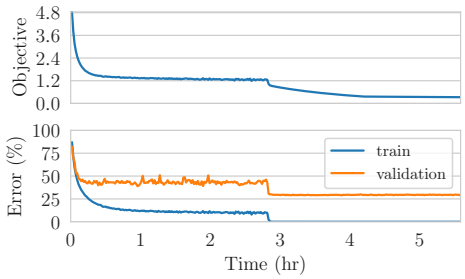


Figure C.11: *Convergence plot of SGD on CIFAR 100 with DN architecture.*

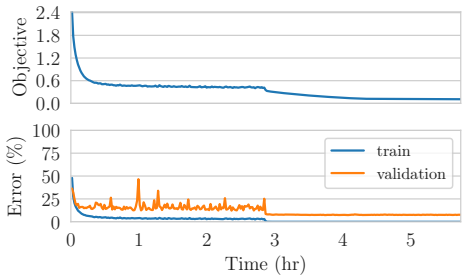


Figure C.12: *Convergence plot of SGD on CIFAR 10 with DN architecture.*

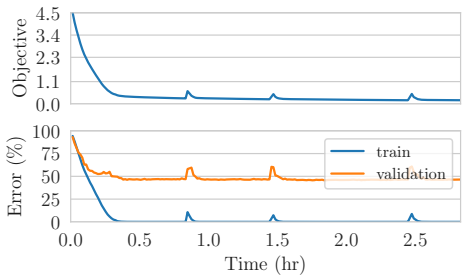


Figure C.13: *Convergence plot of Adagrad on CIFAR 100 with WRN architecture.*

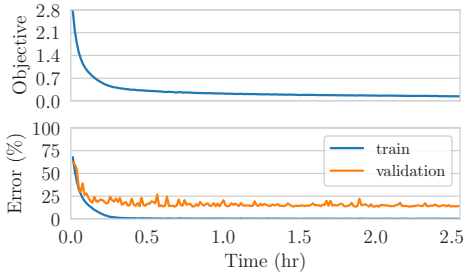


Figure C.14: *Convergence plot of Adagrad on CIFAR 10 with WRN architecture.*

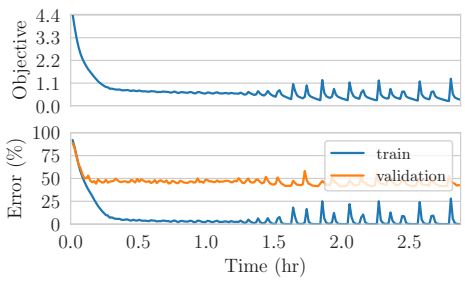


Figure C.15: *Convergence plot of Adam on CIFAR 100 with WRN architecture.*

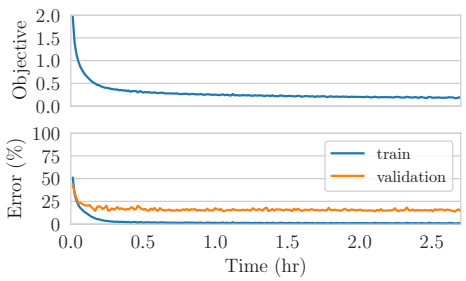


Figure C.16: *Convergence plot of Adam on CIFAR 10 with WRN architecture.*

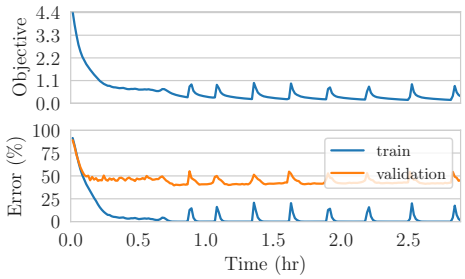


Figure C.17: *Convergence plot of AMSGrad on CIFAR 100 with WRN architecture.*

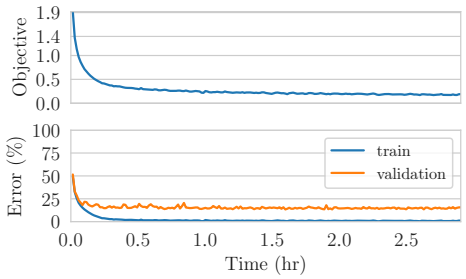


Figure C.18: *Convergence plot of AMSGrad on CIFAR 10 with WRN architecture.*

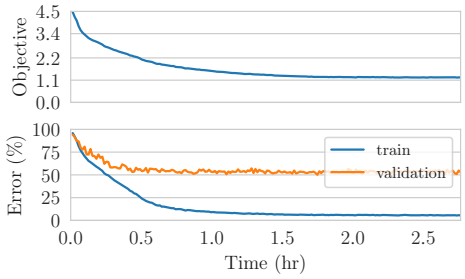


Figure C.19: *Convergence plot of BPGrad on CIFAR 100 with WRN architecture.*

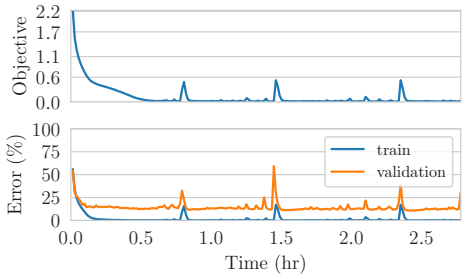


Figure C.20: *Convergence plot of BPGrad on CIFAR 10 with WRN architecture.*

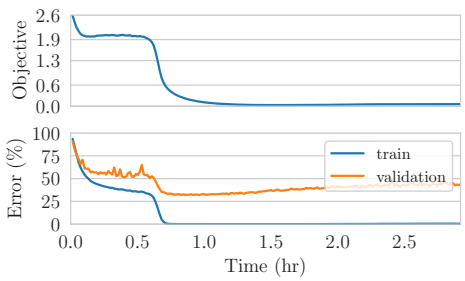


Figure C.21: *Convergence plot of DFW on CIFAR 100 with WRN architecture.*

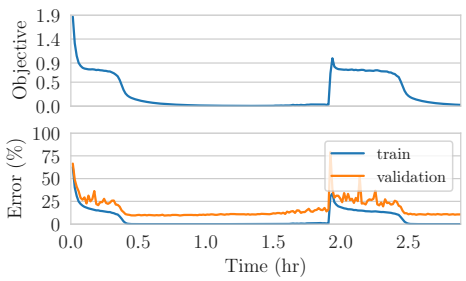


Figure C.22: *Convergence plot of DFW on CIFAR 10 with WRN architecture.*

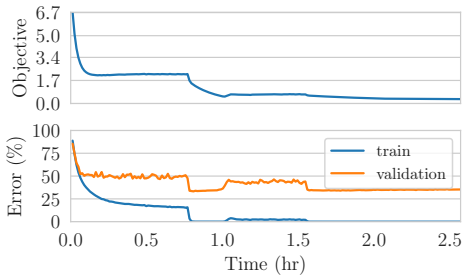


Figure C.23: *Convergence plot of SGD on CIFAR 100 with WRN architecture.*

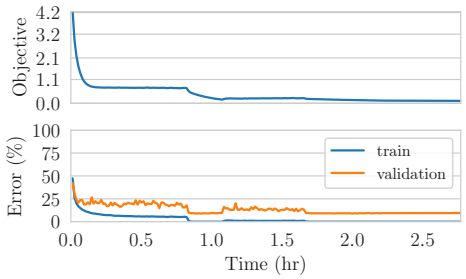


Figure C.24: *Convergence plot of SGD on CIFAR 10 with WRN architecture.*

C.2.4 SGD & DFW: Sensitivity Analysis

We provide here a sensitivity analysis of the DFW algorithm on its hyper-parameter η, and we compare it against the SGD algorithm with its custom schedule. These experiments do not use data augmentation.

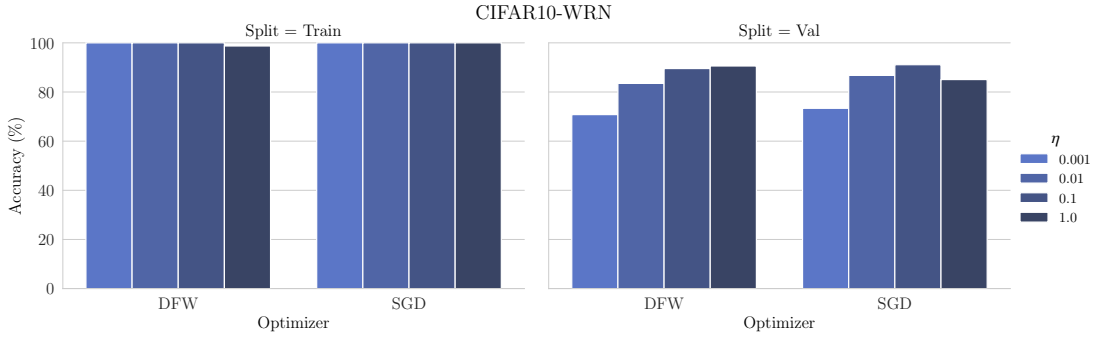


Figure C.25: *Sensitivity analysis on the WRN architecture and CIFAR-10 data set.*

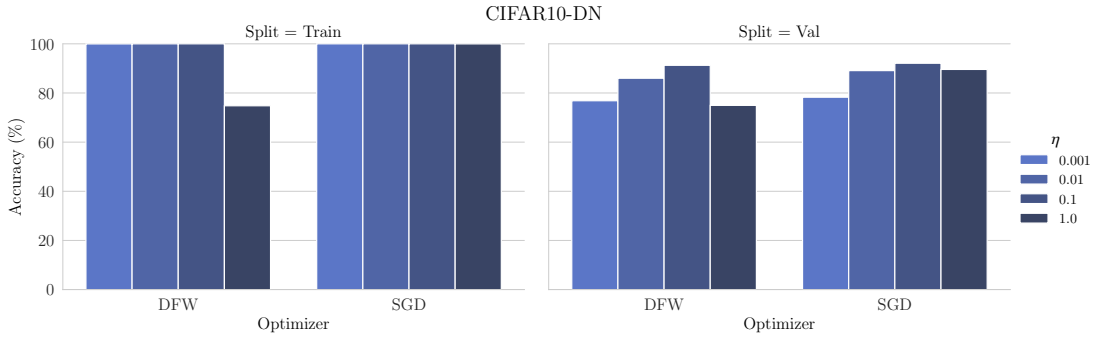


Figure C.26: *Sensitivity analysis on the DN architecture and CIFAR-10 data set.*

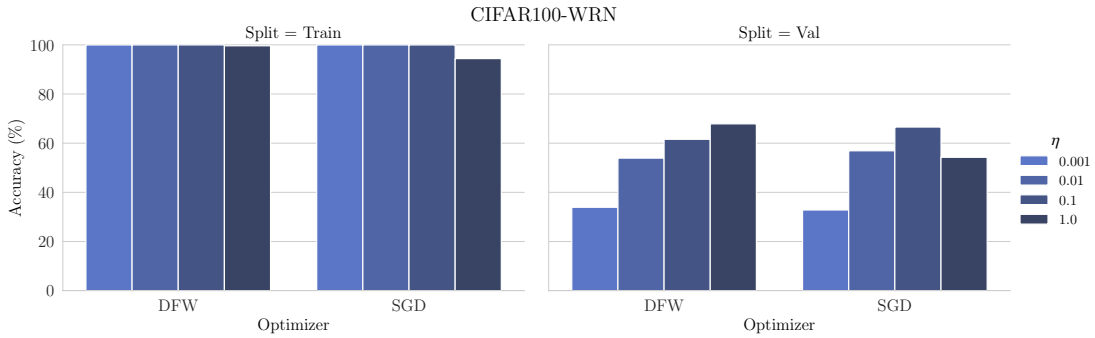


Figure C.27: *Sensitivity analysis on the WRN architecture and CIFAR-100 data set.*

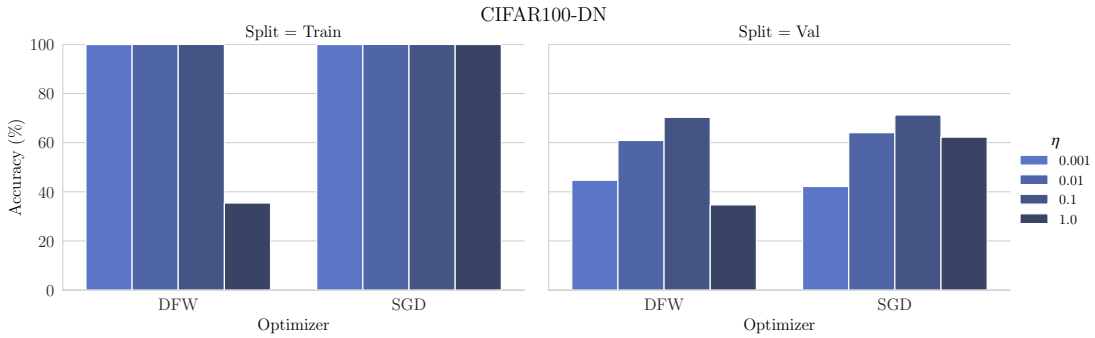


Figure C.28: *Sensitivity analysis on the DN architecture and CIFAR-100 data set.*

C.3 Experimental Details on the SNLI Data Set

C.3.1 Cross-Validation

η	Accuracy CE (%)	Accuracy SVM (%)
0.001	83.43	84.16
0.01	**83.77**	**84.62**
0.1	62.09	34.5

Table C.13: Cross-Validation for ADAGRAD on BiLSTM architecture (best validation accuracy obtained during training).

η	Accuracy CE (%)	Accuracy SVM (%)
1e-05	83.18	83.02
0.0001	**84.56**	**84.69**
0.001	84.42	83.31
0.01	33.82	33.82

Table C.14: Cross-Validation for ADAM on BiLSTM architecture (best validation accuracy obtained during training).

η	Accuracy CE (%)	Accuracy SVM (%)
1e-05	82.81	82.95
0.0001	**84.69**	**84.83**
0.001	84.66	83.59
0.01	36.78	38.25

Table C.15: Cross-Validation for AMSGRAD on BiLSTM architecture (best validation accuracy obtained during training).

η	Accuracy CE (%)	Accuracy SVM (%)
0.001	75.51	74.87
0.01	83.09	83.02
0.1	83.93	84.24
1.0	**84.28**	**84.73**
10	33.82	33.31

Table C.16: Cross-Validation for BPGRAD on BiLSTM architecture (best validation accuracy obtained during training).

η	Accuracy (%)
0.1	84.87
1.0	**85.21**
10	84.76

Table C.17: Cross-Validation for DFW on BiLSTM architecture (best validation accuracy obtained during training).

η	Accuracy CE (%)	Accuracy SVM (%)
0.01	84.22	84.59
0.1	84.63	**85.15**
1.0	**85.06**	84.7
10	34.59	34.51

Table C.18: Cross-Validation for SGD on BiLSTM architecture (best validation accuracy obtained during training).

D

Training Neural Networks for and by Interpolation

Contents

D.1 Local Interpretation of the Polyak Step-Size

Proposition 18. *Suppose that the problem is unconstrained: $\Omega = \mathbb{R}^p$. Let $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{f(\boldsymbol{w}_t) - f_\star}{\|\nabla f(\boldsymbol{w}_t)\|^2} \nabla f(\boldsymbol{w}_t)$. Then $\boldsymbol{w}_{t+1}$ verifies:*

$$\boldsymbol{w}_{t+1} = \operatorname*{argmin}_{\boldsymbol{w} \in \mathbb{R}^p} \|\boldsymbol{w} - \boldsymbol{w}_t\| \text{ subject to: } f(\boldsymbol{w}_t) + \nabla f(\boldsymbol{w}_t)^\top (\boldsymbol{w} - \boldsymbol{w}_t) = f_\star, \quad \text{(D.1)}$$

where we remind that $f_\star$ is the minimum of f, and $\boldsymbol{w} \mapsto f(\boldsymbol{w}_t) + \nabla f(\boldsymbol{w}_t)^\top (\boldsymbol{w} - \boldsymbol{w}_t)$ is the linearization of f at $\boldsymbol{w}_t$. In other words, $\boldsymbol{w}_{t+1}$ is the closest point to $\boldsymbol{w}_t$ that lies on the hyper-plane $f(\boldsymbol{w}_t) + \nabla f(\boldsymbol{w}_t)^\top (\boldsymbol{w} - \boldsymbol{w}_t) = f_\star$.

Proof. See Appendix D.4.1 □

D.2 Convergence Results

Before we detail our convergence results, we introduce the notions of uniform lower bound and ε-interpolation.

D.2.1 Notation

Intuitively, a uniform lower bound on the problem $(\mathcal{P})$ is a lower bound on all loss functions ℓ_z on their unconstrained domain $\mathbb{R}^p$. We formalize this below:

Definition 3 (Uniform Lower Bound)**.** *We say that $\underline{B}$ is a uniform lower bound on $(\mathcal{P})$ if:*

$$\underline{B} \leq \inf_{z \in \mathcal{Z}} \inf_{\boldsymbol{w} \in \mathbb{R}^p} \ell_z(\boldsymbol{w}). \quad \text{(D.2)}$$

Note that in the main paper, we have used the special case $\underline{\mathrm{B}} = 0$ as the uniform lower bound. The definition above makes $\underline{\mathrm{B}}$ a useful statistic to analyze the behavior of each loss function ℓ_z around $\mathbf{w}_\star$, in a uniform way (that is, independently of z). The quality of a uniform lower bound $\underline{\mathrm{B}}$ can be quantified by the notion of ε-interpolation:

Definition 4 (ε-Interpolation)**.** *Let $\underline{B}$ be a uniform lower bound on $(\mathcal{P})$, $\mathbf{w}_\star$ be a solution of $(\mathcal{P})$ and $\varepsilon \geq 0$ be a non-negative number. Then we say that $\mathbf{w}_\star$ is an ε-interpolation for $((\mathcal{P}), \underline{B})$ if:*

$$\forall z \in \mathcal{Z},\ \ell_z(\mathbf{w}_\star) - \underline{B} \leq \varepsilon. \quad \text{(D.3)}$$

By taking the expectation over equation (D.3), we can see that if $\mathbf{w}_\star$ is an ε-interpolation for $((\mathcal{P}), \underline{\mathrm{B}})$, then we immediately have: $f_\star - \varepsilon \leq \underline{\mathrm{B}} \leq f_\star$. In other words, $\underline{\mathrm{B}}$ is also an approximation of $f_\star$ by below, and its quality is quantified by ε. We further note that $f_\star$ does not satisfy the definition of a uniform lower bound in the general case. However when $f_\star$ actually is a uniform lower bound, for any solution $\mathbf{w}_\star$ of $(\mathcal{P})$, $\mathbf{w}_\star$ is a $\varepsilon = 0$-interpolation for $((\mathcal{P}),\ f_\star)$.

In the general case where $\underline{\mathrm{B}}$ may be different from 0, the ALI-G step-size can be defined as:

$$\gamma_t = \min\left\{\frac{\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 + \delta}, \eta\right\} \tag{D.4}$$

We now turn to our convergence results, which give convergence rates in three settings: convex and Lipschitz functions, convex and smooth function, and strongly convex and smooth functions. In each setting, we analyze three regimes which complement each other: no (infinite) maximal learning-rate, large maximal learning-rate and small maximal learning-rate.

D.2.2 Lipschitz Convex Functions

Theorem 3. *[Convex and Lipschitz] We assume that $\mathcal{X}$ is a convex set, and that for every $z \in \mathcal{Z}$, ℓ_z is convex and C-Lipschitz. Let $\underline{\mathrm{B}}$ be a uniform lower bound on $(\mathcal{P})$ and $\mathbf{w}_\star$ be a solution of $(\mathcal{P})$. Further suppose that $\mathbf{w}_\star$ is an ε-interpolation for $((\mathcal{P}), \underline{\mathrm{B}})$. Then ALI-G$^\infty$ applied to f satisfies:*

$$f\left(\frac{1}{T+1}\sum_{t=0}^{T}\boldsymbol{w}_t\right) - f_\star \leq \varepsilon\sqrt{\left(\frac{C^2}{\delta}+1\right)} + \frac{\|\boldsymbol{w}_0 - \mathbf{w}_\star\|\sqrt{C^2+\delta}}{\sqrt{T+1}}. \tag{D.5}$$

Proof. See Appendix D.4.3. □

Theorem 4. *We assume that $\mathcal{X}$ is a convex set, and that for every $z \in \mathcal{Z}$, ℓ_z is convex and C-Lipschitz. Let $\mathbf{w}_\star$ be an ε-interpolation for $((\mathcal{P}), \underline{\mathrm{B}})$. We further assume that $\eta > \frac{\varepsilon}{\delta}$. Then if we apply ALI-G with a maximal learning-rate of η to f,*

we have:

$$f\left(\frac{1}{T+1}\sum_{t=0}^{T}\boldsymbol{w}_t\right)-f_\star\leq\frac{\|\boldsymbol{w}_0-\mathbf{w}_\star\|^2}{(\eta-\frac{\varepsilon}{\delta})(T+1)}+\frac{\varepsilon^2}{\delta(\eta-\frac{\varepsilon}{\delta})}+\sqrt{\frac{(C^2+\delta)\|\boldsymbol{w}_0-\mathbf{w}_\star\|^2}{T+1}}+\varepsilon\sqrt{\frac{C^2}{\delta}+1}. \tag{D.6}$$

Proof. See Appendix D.4.4. □

We note that for very large values of η ($\eta\to\infty$), Theorem 4 gives the exact same result as Theorem 3. However when η is small, the convergence error of Theorem 4 is large. This is corrected in the following result which is informative in the regime where η is small:

Theorem 5. *We assume that $\mathcal{X}$ is a convex set, and that for every $z\in\mathcal{Z}$, ℓ_z is convex and C-Lipschitz. Let $\mathbf{w}_\star$ be an ε-interpolation for $((\mathcal{P}),\underline{B})$. Then if we apply ALI-G with a maximal learning-rate of η to f, we have:*

$$f\left(\frac{1}{T+1}\sum_{t=0}^{T}\boldsymbol{w}_t\right)-f_\star\leq\frac{\|\boldsymbol{w}_0-\mathbf{w}_\star\|^2}{\eta(T+1)}+\varepsilon+\sqrt{\frac{(C^2+\delta)\|\boldsymbol{w}_0-\mathbf{w}_\star\|^2}{T+1}}+\eta\varepsilon\sqrt{C^2+\delta}. \tag{D.7}$$

Proof. See Appendix D.4.5. □

D.2.3 Smooth Convex Functions

We now tackle the convex and β-smooth case. Our proof techniques naturally produce the separation $\eta\geq\frac{1}{2\beta}$ and $\eta\leq\frac{1}{2\beta}$. Whenever $\eta\geq\frac{1}{2\beta}$, the convergence result is exactly the same as when $\eta\to\infty$. When $\eta\leq\frac{1}{2\beta}$, the speed of convergence is limited by the value of η.

Theorem 6. *[Convex and Smooth] We assume that $\mathcal{X}$ is a convex set, and that for every $z\in\mathcal{Z}$, ℓ_z is convex and β-smooth. Let $\underline{B}$ be a uniform lower bound on $(\mathcal{P})$ and $\mathbf{w}_\star$ be a solution of $(\mathcal{P})$. Further suppose that $\mathbf{w}_\star$ is an ε-interpolation for $((\mathcal{P}),\underline{B})$, and that $\delta>2\beta\varepsilon$. Then ALI-G$^\infty$ applied to f satisfies:*

$$f\left(\frac{1}{T+1}\sum_{t=0}^{T}\boldsymbol{w}_t\right)-f_\star\leq\frac{\delta}{\beta(1-\frac{2\beta\varepsilon}{\delta})}+\frac{2\beta}{1-\frac{2\beta\varepsilon}{\delta}}\frac{\|\boldsymbol{w}_0-\mathbf{w}_\star\|^2}{T+1}. \tag{D.8}$$

Proof. See Appendix D.4.6. □

Theorem 7. *We assume that $\mathcal{X}$ is a convex set, and that for every $z \in \mathcal{Z}$, ℓ_z is convex and β-smooth. Let $\mathbf{w}_\star$ be an ε-interpolation for $((\mathcal{P}), \underline{B})$, and suppose that $\delta > 2\beta\varepsilon$. Further assume that $\eta \geq \frac{1}{2\beta}$. Then if we apply ALI-G with a maximal learning-rate of η to f, we have:*

$$f\left(\frac{1}{T+1}\sum_{t=0}^{T}\boldsymbol{w}_t\right) - f_\star \leq \frac{\delta}{\beta(1-\frac{2\beta\varepsilon}{\delta})} + \frac{2\beta}{1-\frac{2\beta\varepsilon}{\delta}}\frac{\|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2}{T+1}. \tag{D.9}$$

Proof. See Appendix D.4.7. □

Theorem 8. *We assume that $\mathcal{X}$ is a convex set, and that for every $z \in \mathcal{Z}$, ℓ_z is convex and β-smooth. Let $\mathbf{w}_\star$ be an ε-interpolation for $((\mathcal{P}), \underline{B})$, and suppose that $\delta > 2\beta\varepsilon$. Further assume that $\eta \leq \frac{1}{2\beta}$. Then if we apply ALI-G with a maximal learning-rate of η to f, we have:*

$$f\left(\frac{1}{T+1}\sum_{t=0}^{T}\boldsymbol{w}_t\right) - f_\star \leq \frac{\|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2}{\eta(T+1)} + \frac{\delta}{2\beta} + \varepsilon. \tag{D.10}$$

Proof. See Appendix D.4.8. □

D.2.4 Smooth and Strongly Convex Functions

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 $\eta \leq \frac{1}{2\beta}$. In a similar way to the β-smooth case, when $\eta \geq \frac{1}{2\beta}$, Theorem 10 gives the exact same result as $\eta \to \infty$. And when $\eta \leq \frac{1}{2\beta}$, the rate of convergence given by Theorem 11 is limited by the value of η.

Theorem 9. *[Strongly Convex and Smooth] We assume that $\mathcal{X}$ is a convex set, and that for every $z \in \mathcal{Z}$, ℓ_z is α-strongly convex and β-smooth. Let $\underline{B}$ be a uniform lower bound on $(\mathcal{P})$ and $\mathbf{w}_\star$ be a solution of $(\mathcal{P})$. Further suppose that $\mathbf{w}_\star$ is an ε-interpolation for $((\mathcal{P}), \underline{B})$, and that $\delta > 2\beta\varepsilon$. Then ALI-G$^\infty$ applied to f satisfies:*

$$f(\boldsymbol{w}_{T+1}) - f_\star \leq \beta \exp\left(-\frac{\alpha T}{8\beta}\right)\|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 + \frac{2\delta}{\alpha} + \left(10\frac{\beta}{\alpha} + 4\frac{\beta^2}{\alpha^2}\right)\varepsilon. \tag{D.11}$$

In other words, f approximately converges to $f_\star$ at a rate of $\mathcal{O}(\exp(-\alpha T/8\beta))$.

Proof. See Appendix D.4.9. □

Theorem 10. *We assume that $\mathcal{X}$ is a convex set, and that for every $z \in \mathcal{Z}$, ℓ_z is α-strongly convex and β-smooth. Let $\mathbf{w}_\star$ be an ε-interpolation for $((\mathcal{P}), \underline{B})$, and suppose that $\delta > 2\beta\varepsilon$. Further assume that $\eta \geq \frac{1}{2\beta}$. Then if we apply ALI-G with a maximal learning-rate of η to f, we have:*

$$f(\boldsymbol{w}_{T+1}) - f_\star \leq \beta \exp\left(-\frac{\alpha T}{8\beta}\right) \|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 + \frac{2\delta}{\alpha} + \left(10\frac{\beta}{\alpha} + 4\frac{\beta^2}{\alpha^2}\right)\varepsilon. \tag{D.12}$$

Proof. See Appendix D.4.10. □

Theorem 11. *We assume that $\mathcal{X}$ is a convex set, and that for every $z \in \mathcal{Z}$, ℓ_z is α-strongly convex and β-smooth. Let $\mathbf{w}_\star$ be an ε-interpolation for $((\mathcal{P}), \underline{B})$, and suppose that $\delta > 2\beta\varepsilon$. Further assume that $\eta \leq \frac{1}{2\beta}$. Then if we apply ALI-G with a maximal learning-rate of η to f, we have:*

$$f(\boldsymbol{w}_{T+1}) - f_\star \leq \beta \exp\left(-\frac{\alpha\eta T}{4}\right) \|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 + \frac{2\delta}{\alpha} + \frac{14\varepsilon\beta}{\alpha}. \tag{D.13}$$

Proof. See Appendix D.4.11. □

D.3 On the Need for a Maximal Learning-Rate for Non-Convexity

The Restricted Secant Inequality (RSI) is a milder assumption than convexity. It can be defined as follows:

Definition 5. *Let $f : \mathbb{R}^p \to \mathbb{R}$ be a lower-bounded differentiable function achieving its minimum at $\mathbf{w}_\star$. We say that f satisfies the RSI if there exists $\alpha > 0$ such that:*

$$\forall \boldsymbol{w} \in \mathbb{R}^p,\ \nabla f(\boldsymbol{w})^\top (\boldsymbol{w} - \mathbf{w}_\star) \geq \alpha \|\boldsymbol{w} - \mathbf{w}_\star\|^2. \tag{D.14}$$

The RSI is sometimes used to prove convergence of optimization algorithms without assuming convexity (Vaswani et al., 2019b).

As we prove below, the Polyak step-size may not convergence under the RSI assumption, even in a non-stochastic setting with the exact minimum known.

We introduce the function $f : w \in [\frac{-3}{5}; \frac{3}{5}] \mapsto w^2 - |w|^3$. We restrict our domain of study to $[\frac{-3}{5}; \frac{3}{5}]$ for simplicity purposes – an extension to $\mathbb{R}$ can easily be constructed

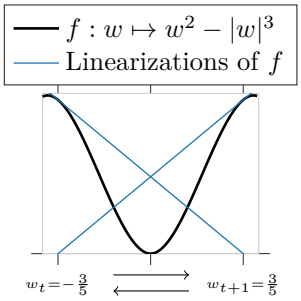


Figure D.1: *Illustration of the function f, which satisfies the RSI. When starting at $w = -3/5$, gradient descent with the Polyak step-size oscillates between $w = -3/5$ and $w = 3/5$.*

by extending f. We will first show that f fulfills the RSI assumption, and then that it oscillates between two points for a well chosen initialization.

Let us show that f satisfies the RSI with $\alpha = \frac{1}{5}$. First we note that f achieves its minimum at $w_\star = 0$, and that $f(w_\star) = 0$. In addition, we introduce the sign function $\sigma(w)$, which is equal to 1 if $w \geq 0$, and -1 otherwise. Now let $w \in [\frac{-3}{5}; \frac{3}{5}]$. Then we have that:

$$\begin{aligned} \nabla f(w)(w - w_\star) - \frac{1}{5}(w - w_\star)^2 &= (2w - 3\sigma(w)w^2)(w - 0) - \frac{1}{5}(w - 0)^2, \\ &= \frac{9}{5}w^2 - 3\sigma(w)w^3, \\ &= 3w^2(\frac{3}{5} - \sigma(w)w), \\ &\geq 0. \end{aligned} \tag{D.15}$$

Now let us show that if we apply gradient descent with a Polyak step-size to f, with starting point $w_0 = \frac{-3}{5}$, we obtain $w_1 = \frac{3}{5}$. This will prove oscillation of the iterates by symmetry of the problem. Let $w_0 = \frac{-3}{5}$. Then we have $f(w_0) = \frac{9}{25} - \frac{27}{125} = \frac{18}{125}$. Furthermore, $\nabla f(w_0) = 2(\frac{-3}{5}) + 3(\frac{9}{25}) = \frac{-3}{25}$. Therefore:

$$\begin{aligned} w_1 &= w_0 - \frac{f(w_0) - 0}{(\nabla f(w_0))^2}\nabla f(w_0), \\ &= w_0 - \frac{f(w_0)}{\nabla f(w_0)}, \\ &= \frac{-3}{5} + \frac{\frac{18}{125}}{\frac{3}{25}}, \\ &= \frac{-3}{5} + \frac{6}{5}, \\ &= \frac{3}{5}. \end{aligned} \tag{D.16}$$

D.4 Proofs

D.4.1 Proposition 18

Proposition 18. *Suppose that the problem is unconstrained: $\Omega = \mathbb{R}^p$. Let $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{f(\boldsymbol{w}_t)-f_\star}{\|\nabla f(\boldsymbol{w}_t)\|^2}\nabla f(\boldsymbol{w}_t)$. Then $\boldsymbol{w}_{t+1}$ verifies:*

$$\boldsymbol{w}_{t+1} = \operatorname*{argmin}_{\boldsymbol{w}\in\mathbb{R}^p} \|\boldsymbol{w} - \boldsymbol{w}_t\| \text{ subject to: } f(\boldsymbol{w}_t) + \nabla f(\boldsymbol{w}_t)^\top(\boldsymbol{w} - \boldsymbol{w}_t) = f_\star, \quad \text{(D.1)}$$

where we remind that $f_\star$ is the minimum of f, and $\boldsymbol{w} \mapsto f(\boldsymbol{w}_t) + \nabla f(\boldsymbol{w}_t)^\top(\boldsymbol{w} - \boldsymbol{w}_t)$ is the linearization of f at $\boldsymbol{w}_t$. In other words, $\boldsymbol{w}_{t+1}$ is the closest point to $\boldsymbol{w}_t$ that lies on the hyper-plane $f(\boldsymbol{w}_t) + \nabla f(\boldsymbol{w}_t)^\top(\boldsymbol{w} - \boldsymbol{w}_t) = f_\star$.

Proof. First we show that $\boldsymbol{w}_{t+1}$ satisfies the linear equality constraint:

$$\begin{aligned} &f(\boldsymbol{w}_t) + \nabla f(\boldsymbol{w}_t)^\top(\boldsymbol{w}_{t+1} - \boldsymbol{w}_t) \\ &\quad = f(\boldsymbol{w}_t) + \nabla f(\boldsymbol{w}_t)^\top \left(-\frac{f(\boldsymbol{w}_t) - f_\star}{\|\nabla f(\boldsymbol{w}_t)\|^2}\nabla f(\boldsymbol{w}_t)\right), \\ &\quad = f(\boldsymbol{w}_t) - f(\boldsymbol{w}_t) + f_\star, \\ &\quad = f_\star. \end{aligned} \quad \text{(D.17)}$$

Now let us show that it has a minimal distance to $\boldsymbol{w}_t$.

We take $\hat{\boldsymbol{w}} \in \mathbb{R}^p$ a solution of the linear equality constraint, and we will show that $\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_t\| \leq \|\hat{\boldsymbol{w}} - \boldsymbol{w}_t\|$. By definition, we have that $\hat{\boldsymbol{w}}$ satisfies:

$$f(\boldsymbol{w}_t) + \nabla f(\boldsymbol{w}_t)^\top(\hat{\boldsymbol{w}} - \boldsymbol{w}_t) = f_\star. \quad \text{(D.18)}$$

Now we can write:

$$\begin{aligned} \|\boldsymbol{w}_{t+1} - \boldsymbol{w}_t\| &= \|\frac{f(\boldsymbol{w}_t) - f_\star}{\|\nabla f(\boldsymbol{w}_t)\|^2}\nabla f(\boldsymbol{w}_t)\|, \\ &= \frac{f(\boldsymbol{w}_t) - f_\star}{\|\nabla f(\boldsymbol{w}_t)\|}, \\ &= \frac{|\nabla f(\boldsymbol{w}_t)^\top(\hat{\boldsymbol{w}} - \boldsymbol{w}_t)|}{\|\nabla f(\boldsymbol{w}_t)\|}, \\ &\leq \frac{\|\nabla f(\boldsymbol{w}_t)\|\|\hat{\boldsymbol{w}} - \boldsymbol{w}_t\|}{\|\nabla f(\boldsymbol{w}_t)\|}, \quad \text{(Cauchy-Schwarz)} \\ &= \|\hat{\boldsymbol{w}} - \boldsymbol{w}_t\|. \end{aligned} \quad \text{(D.19)}$$

□

D.4.2 Theorem 8

Proposition 8. *[Proximal Interpretation] Suppose that $\Omega = \mathbb{R}^p$ and let $\delta = 0$. We consider the update performed by SGD: $\boldsymbol{w}_{t+1}^{SGD} = \boldsymbol{w}_t - \eta_t \nabla \ell_{z_t}(\boldsymbol{w}_t)$; and the update performed by ALI-G: $\boldsymbol{w}_{t+1}^{ALI\text{-}G} = \boldsymbol{w}_t - \gamma_t \nabla \ell_{z_t}(\boldsymbol{w}_t)$, where $\gamma_t = \min\left\{\frac{\ell_{z_t}(\boldsymbol{w}_t)}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 + \delta}, \eta\right\}$. Then we have:*

$$\boldsymbol{w}_{t+1}^{SGD} = \operatorname*{argmin}_{\boldsymbol{w}\in\mathbb{R}^p} \left\{ \frac{1}{2\eta_t}\|\boldsymbol{w} - \boldsymbol{w}_t\|^2 + \ell_{z_t}(\boldsymbol{w}_t) + \nabla \ell_{z_t}(\boldsymbol{w}_t)^\top (\boldsymbol{w} - \boldsymbol{w}_t) \right\}, \tag{5.4}$$

$$\boldsymbol{w}_{t+1}^{ALI\text{-}G} = \operatorname*{argmin}_{\boldsymbol{w}\in\mathbb{R}^p} \left\{ \frac{1}{2\eta}\|\boldsymbol{w} - \boldsymbol{w}_t\|^2 + \max\left\{ \ell_{z_t}(\boldsymbol{w}_t) + \nabla \ell_{z_t}(\boldsymbol{w}_t)^\top (\boldsymbol{w} - \boldsymbol{w}_t), 0 \right\} \right\}. \tag{5.5}$$

Proof. We tackle the slightly more general case where $\underline{\mathrm{B}}$ may be different from zero. In order to make the notation simpler, we use $\boldsymbol{d}_t \triangleq \nabla \ell_{z_t}(\boldsymbol{w}_t)$ and $l_t \triangleq \ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}}$. First, let us consider $\boldsymbol{d}_t = \boldsymbol{0}$.

> Then we choose $\gamma_t = 0$ and it is clear that $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta\gamma_t \boldsymbol{d}_t = \boldsymbol{w}_t$ is the optimal solution of problem (5.5).

We now assume $\boldsymbol{d}_t \neq \boldsymbol{0}$.

> We can successively re-write the proximal problem (5.5) as :
>
> $$\begin{aligned}
> &\min_{\boldsymbol{w}\in\mathbb{R}^p} \left\{ \frac{1}{2\eta}\|\boldsymbol{w} - \boldsymbol{w}_t\|^2 + \max\left\{ \ell_{z_t}(\boldsymbol{w}_t) + \nabla \ell_{z_t}(\boldsymbol{w}_t)^\top (\boldsymbol{w} - \boldsymbol{w}_t), \underline{\mathrm{B}} \right\} \right\}, \\
> &\min_{\boldsymbol{w}\in\mathbb{R}^p} \left\{ \frac{1}{2\eta}\|\boldsymbol{w} - \boldsymbol{w}_t\|^2 + \max\left\{ \ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}} + \nabla \ell_{z_t}(\boldsymbol{w}_t)^\top (\boldsymbol{w} - \boldsymbol{w}_t), 0 \right\} \right\}, \\
> &\min_{\boldsymbol{w}\in\mathbb{R}^p} \left\{ \frac{1}{2\eta}\|\boldsymbol{w} - \boldsymbol{w}_t\|^2 + \max\left\{ l_t + \boldsymbol{d}_t^\top (\boldsymbol{w} - \boldsymbol{w}_t), 0 \right\} \right\}, \\
> &\min_{\boldsymbol{w}\in\mathbb{R}^p, \upsilon} \left\{ \frac{1}{2\eta}\|\boldsymbol{w} - \boldsymbol{w}_t\|^2 + \upsilon \right\} \text{ subject to: } \upsilon \geq 0,\ \upsilon \geq l_t + \boldsymbol{d}_t^\top (\boldsymbol{w} - \boldsymbol{w}_t) \\
> &\min_{\boldsymbol{w}\in\mathbb{R}^p, \upsilon} \sup_{\mu,\nu\geq 0} \left\{ \frac{1}{2\eta}\|\boldsymbol{w} - \boldsymbol{w}_t\|^2 + \upsilon - \mu\upsilon - \nu(\upsilon - l_t - \boldsymbol{d}_t^\top (\boldsymbol{w} - \boldsymbol{w}_t)) \right\} \\
> &\sup_{\mu,\nu\geq 0} \min_{\boldsymbol{w}\in\mathbb{R}^p, \upsilon} \left\{ \frac{1}{2\eta}\|\boldsymbol{w} - \boldsymbol{w}_t\|^2 + \upsilon - \mu\upsilon - \nu(\upsilon - l_t - \boldsymbol{d}_t^\top (\boldsymbol{w} - \boldsymbol{w}_t)) \right\}, \qquad \text{(D.20)}
> \end{aligned}$$
>
> where the last equation uses strong duality. The inner problem is now smooth in $\boldsymbol{w}$ and υ. We write its KKT conditions:
>
> $$\frac{\partial \cdot}{\partial \upsilon} = 0 : \quad 1 - \mu - \nu = 0 \tag{D.21}$$

$$\frac{\partial \cdot}{\partial \boldsymbol{w}} = 0 : \quad \frac{1}{\eta}(\boldsymbol{w} - \boldsymbol{w}_t) + \nu \boldsymbol{d}_t = \mathbf{0} \tag{D.22}$$

We plug in these results and obtain:

$$\begin{aligned} &\sup_{\mu,\nu \geq 0} \left\{ \frac{1}{2\eta} \|\eta\nu \boldsymbol{d}_t\|^2 + \nu(l_t + \boldsymbol{d}_t^\top(-\eta\nu \boldsymbol{d}_t)) \right\} \\ &\text{st:} \quad \mu + \nu = 1 \\ &\sup_{\nu \in [0,1]} \left\{ \frac{\eta}{2}\nu^2 \|\boldsymbol{d}_t\|^2 + \nu l_t - \eta\nu^2 \|\boldsymbol{d}_t^\top\|^2 \right\} \\ &\sup_{\nu \in [0,1]} \left\{ -\frac{\eta}{2}\nu^2 \|\boldsymbol{d}_t\|^2 + \nu l_t \right\} \end{aligned} \tag{D.23}$$

This is a one-dimensional quadratic problem in ν. It can be solved in closed-form by finding the global maximum of the quadratic objective, and projecting the solution on $[0, 1]$. We have:

$$\frac{\partial \cdot}{\partial \nu} = 0 : -\eta\nu \|\boldsymbol{d}_t\|^2 + l_t = 0 \tag{D.24}$$

Since $\boldsymbol{d}_t \neq \mathbf{0}$ and $\eta \neq 0$, this gives the optimal solution:

$$\nu = \min \left\{ \max \left\{ \frac{l_t}{\eta \|\boldsymbol{d}_t\|^2}, 0 \right\}, 1 \right\} = \min \left\{ \frac{l_t}{\eta \|\boldsymbol{d}_t\|^2}, 1 \right\}, \tag{D.25}$$

since $l_t, \eta, \|\boldsymbol{d}_t\|^2 \geq 0$.
Plugging this back in the KKT conditions, we obtain that the solution $\boldsymbol{w}_{t+1}$ of the primal problem can be written as:

$$\begin{aligned} \boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \eta\nu \boldsymbol{d}_t, \\ &= \boldsymbol{w}_t - \eta \min \left\{ \frac{l_t}{\eta \|\boldsymbol{d}_t\|^2}, 1 \right\} \boldsymbol{d}_t, \\ &= \boldsymbol{w}_t - \eta \min \left\{ \frac{\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}}}{\eta \|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2}, 1 \right\} \nabla \ell_{z_t}(\boldsymbol{w}_t), \\ &= \boldsymbol{w}_t - \min \left\{ \frac{\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2}, \eta \right\} \nabla \ell_{z_t}(\boldsymbol{w}_t). \end{aligned} \tag{D.26}$$

□

D.4.3 Theorem 3

Theorem 3. *[Convex and Lipschitz] We assume that $\mathcal{X}$ is a convex set, and that for every $z \in \mathcal{Z}$, ℓ_z is convex and C-Lipschitz. Let $\underline{\mathrm{B}}$ be a uniform lower bound on $(\mathcal{P})$ and $\mathbf{w}_\star$ be a solution of $(\mathcal{P})$. Further suppose that $\mathbf{w}_\star$ is an ε-interpolation for*

((P), $\underline{B}$). Then ALI-G$^\infty$ applied to f satisfies:

$$f\left(\frac{1}{T+1}\sum_{t=0}^{T}\boldsymbol{w}_t\right) - f_\star \leq \varepsilon\sqrt{\left(\frac{C^2}{\delta}+1\right)} + \frac{\|\boldsymbol{w}_0 - \mathbf{w}_\star\|\sqrt{C^2+\delta}}{\sqrt{T+1}}. \tag{D.5}$$

Proof.

We consider the update at time t, which we condition on the draw of $z_t \in \mathcal{Z}$:

$$\begin{aligned}
&\|\boldsymbol{w}_{t+1} - \mathbf{w}_\star\|^2 \\
&= \|\Pi_\Omega(\boldsymbol{w}_t - \gamma_t \nabla \ell_{z_t}(\boldsymbol{w}_t)) - \mathbf{w}_\star\|^2 \\
&\leq \|\boldsymbol{w}_t - \gamma_t \nabla \ell_{z_t}(\boldsymbol{w}_t) - \mathbf{w}_\star\|^2 \quad (\Pi_\Omega \text{ projection}) \\
&= \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - 2\gamma_t \nabla \ell_{z_t}(\boldsymbol{w}_t)^\top(\boldsymbol{w}_t - \mathbf{w}_\star) + \gamma_t^2 \|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 \\
&= \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - 2\gamma_t \nabla \ell_{z_t}(\boldsymbol{w}_t)^\top(\boldsymbol{w}_t - \mathbf{w}_\star) + \gamma_t \frac{\ell_{z_t}(\boldsymbol{w}_t) - \underline{B}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 + \delta} \|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 \\
&\qquad \text{(definition of } \gamma_t) \\
&\leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - 2\gamma_t \nabla \ell_{z_t}(\boldsymbol{w}_t)^\top(\boldsymbol{w}_t - \mathbf{w}_\star) + \gamma_t \frac{\ell_{z_t}(\boldsymbol{w}_t) - \underline{B}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2} \|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 \\
&\qquad \text{(because } \ell_{z_t}(\boldsymbol{w}_t) - \underline{B} \geq 0 \text{ and } \delta \geq 0) \\
&\leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - 2\gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \underline{B}) \quad (\text{convexity of } \ell_{z_t}) \\
&= \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \gamma_t\Big((\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) - (\ell_{z_t}(\mathbf{w}_\star) - \underline{B})\Big) \\
&= \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \frac{1}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 + \delta}\Big((\ell_{z_t}(\boldsymbol{w}_t) - \underline{B})(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) \\
&\quad - (\ell_{z_t}(\boldsymbol{w}_t) - \underline{B})(\ell_{z_t}(\mathbf{w}_\star) - \underline{B})\Big) \quad \text{(definition of } \gamma_t) \\
&= \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \frac{1}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 + \delta}\Big((\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star))(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) \\
&\quad + (\ell_{z_t}(\mathbf{w}_\star) - \underline{B})(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) - (\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star))(\ell_{z_t}(\mathbf{w}_\star) - \underline{B}) \\
&\quad - (\ell_{z_t}(\mathbf{w}_\star) - \underline{B})(\ell_{z_t}(\mathbf{w}_\star) - \underline{B})\Big) \\
&\qquad \text{(we use twice } \ell_{z_t}(\boldsymbol{w}_t) - \underline{B} = \ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) + \ell_{z_t}(\mathbf{w}_\star) - \underline{B}) \\
&= \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \frac{1}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 + \delta}\Big((\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star))^2 - (\ell_{z_t}(\mathbf{w}_\star) - \underline{B})^2\Big) \\
&\qquad \text{(middle terms cancel out)} \\
&= \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \frac{(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star))^2}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 + \delta} + \frac{(\ell_{z_t}(\mathbf{w}_\star) - \underline{B})^2}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 + \delta} \\
&\leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \frac{(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star))^2}{C^2 + \delta} + \frac{(\ell_{z_t}(\mathbf{w}_\star) - \underline{B})^2}{\delta} \\
&\qquad \text{(because we have } 0 \leq \|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 \leq C^2) \\
&\leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \frac{(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star))^2}{C^2 + \delta} + \frac{\varepsilon^2}{\delta} \quad (\text{definition of } \varepsilon)
\end{aligned} \tag{D.27}$$

We re-write this inequality as:

$$\left(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)\right)^2 \leq \varepsilon^2 \left(\frac{C^2}{\delta} + 1\right) + \left(C^2 + \delta\right) \left(\|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \|\boldsymbol{w}_{t+1} - \mathbf{w}_\star\|^2\right) \tag{D.28}$$

We can now use the Cauchy-Schwarz inequality to bound the sum over the iterations:

$$(T+1) \sum_{t=0}^{T} \left(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)\right)^2 \geq \left(\sum_{t=0}^{T} \ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)\right)^2 \tag{D.29}$$

Therefore we can write:

$$\begin{aligned}
&\left(\sum_{t=0}^{T} \ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)\right)^2 \\
&\quad \leq (T+1) \sum_{t=0}^{T} \left(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)\right)^2 \\
&\quad \leq (T+1)^2 \varepsilon^2 \left(\frac{C^2}{\delta} + 1\right) + (T+1) \left(C^2 + \delta\right) \left(\|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 - \|\boldsymbol{w}_{T+1} - \mathbf{w}_\star\|^2\right), \\
&\quad \leq (T+1)^2 \varepsilon^2 \left(\frac{C^2}{\delta} + 1\right) + (T+1) \left(C^2 + \delta\right) \|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2,
\end{aligned} \tag{D.30}$$

which yields:

$$\begin{aligned}
&\sum_{t=0}^{T} \ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) \\
&\quad \leq \sqrt{(T+1)^2 \varepsilon^2 \left(\frac{C^2}{\delta} + 1\right) + (T+1) \left(C^2 + \delta\right) \|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2}, \\
&\quad \leq (T+1) \varepsilon \sqrt{\frac{C^2}{\delta} + 1} + \sqrt{(T+1) \left(C^2 + \delta\right) \|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2},
\end{aligned} \tag{D.31}$$

We can now take the expectation over the z_t:

$$\sum_{t=0}^{T} f(\boldsymbol{w}_t) - f_\star \leq (T+1) \varepsilon \sqrt{\frac{C^2}{\delta} + 1} + \sqrt{(T+1) \left(C^2 + \delta\right) \|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2}. \tag{D.32}$$

Dividing by $T+1$ and exploiting convexity of f, we finally get:

$$\begin{aligned} f\left(\frac{1}{T+1}\sum_{t=0}^{T}\boldsymbol{w}_t\right) - f_\star &\leq \frac{1}{T+1}\sum_{t=0}^{T} f(\boldsymbol{w}_t) - f_\star \quad \text{(convexity of } f\text{)} \\ &\leq \varepsilon\sqrt{\frac{C^2}{\delta}+1} + \sqrt{\frac{(C^2+\delta)\left\|\boldsymbol{w}_0 - \mathbf{w}_\star\right\|^2}{T+1}}. \end{aligned} \tag{D.33}$$

□

D.4.4 Theorem 4

Theorem 4. *We assume that $\mathcal{X}$ is a convex set, and that for every $z \in \mathcal{Z}$, ℓ_z is convex and C-Lipschitz. Let $\mathbf{w}_\star$ be an ε-interpolation for ($(\mathcal{P})$, $\underline{\mathrm{B}}$). We further assume that $\eta > \frac{\varepsilon}{\delta}$. Then if we apply ALI-G with a maximal learning-rate of η to f, we have:*

$$\begin{aligned} f\left(\frac{1}{T+1}\sum_{t=0}^{T}\boldsymbol{w}_t\right) - f_\star \leq \frac{\left\|\boldsymbol{w}_0 - \mathbf{w}_\star\right\|^2}{(\eta - \frac{\varepsilon}{\delta})(T+1)} + \frac{\varepsilon^2}{\delta(\eta - \frac{\varepsilon}{\delta})} \\ + \sqrt{\frac{(C^2+\delta)\left\|\boldsymbol{w}_0 - \mathbf{w}_\star\right\|^2}{T+1}} + \varepsilon\sqrt{\frac{C^2}{\delta}+1}. \end{aligned} \tag{D.6}$$

Proof.

We consider the update at time t, which we condition on the draw of $z_t \in \mathcal{Z}$:

$$\begin{aligned} &\left\|\boldsymbol{w}_{t+1} - \mathbf{w}_\star\right\|^2 \\ &\quad= \left\|\Pi_\Omega(\boldsymbol{w}_t - \gamma_t \nabla\ell_{z_t}(\boldsymbol{w}_t)) - \mathbf{w}_\star\right\|^2 \\ &\quad\leq \left\|\boldsymbol{w}_t - \gamma_t \nabla\ell_{z_t}(\boldsymbol{w}_t) - \mathbf{w}_\star\right\|^2 \quad (\Pi_\Omega \text{ projection}) \\ &\quad= \left\|\boldsymbol{w}_t - \mathbf{w}_\star\right\|^2 - 2\gamma_t \nabla\ell_{z_t}(\boldsymbol{w}_t)^\top(\boldsymbol{w}_t - \mathbf{w}_\star) + \gamma_t^2\left\|\nabla\ell_{z_t}(\boldsymbol{w}_t)\right\|^2 \\ &\quad\leq \left\|\boldsymbol{w}_t - \mathbf{w}_\star\right\|^2 - 2\gamma_t \nabla\ell_{z_t}(\boldsymbol{w}_t)^\top(\boldsymbol{w}_t - \mathbf{w}_\star) + \gamma_t \frac{\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}}}{\left\|\nabla\ell_{z_t}(\boldsymbol{w}_t)\right\|^2 + \delta}\left\|\nabla\ell_{z_t}(\boldsymbol{w}_t)\right\|^2 \\ &\qquad\text{(because } \gamma_t \leq \frac{\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}}}{\left\|\nabla\ell_{z_t}(\boldsymbol{w}_t)\right\|^2 + \delta}\text{)} \\ &\quad\leq \left\|\boldsymbol{w}_t - \mathbf{w}_\star\right\|^2 - 2\gamma_t \nabla\ell_{z_t}(\boldsymbol{w}_t)^\top(\boldsymbol{w}_t - \mathbf{w}_\star) + \gamma_t \frac{\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}}}{\left\|\nabla\ell_{z_t}(\boldsymbol{w}_t)\right\|^2}\left\|\nabla\ell_{z_t}(\boldsymbol{w}_t)\right\|^2 \\ &\qquad\text{(because } \ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}} \geq 0 \text{ and } \delta \geq 0\text{)} \\ &\quad\leq \left\|\boldsymbol{w}_t - \mathbf{w}_\star\right\|^2 - 2\gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}}) \quad \text{(convexity of } \ell_{z_t}\text{)} \\ &\quad= \left\|\boldsymbol{w}_t - \mathbf{w}_\star\right\|^2 - 2\gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) \\ &\qquad+ \gamma_t(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}}) \\ &\quad= \left\|\boldsymbol{w}_t - \mathbf{w}_\star\right\|^2 - \gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \gamma_t(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}}) \end{aligned} \tag{D.34}$$

We now consider different cases, according to the value that γ_t takes: $\gamma_t = \frac{\ell_{z_t}(\boldsymbol{w}_t)-\underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta}$ or $\gamma_t = \eta$.

First, suppose that $\gamma_t = \frac{\ell_{z_t}(\boldsymbol{w}_t)-\underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta}$. Then we can follow the proof of Theorem 3 to obtain:

$$\|\boldsymbol{w}_{t+1} - \mathbf{w}_\star\|^2 \leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \frac{(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star))^2}{C^2+\delta} + \frac{\varepsilon^2}{\delta}. \tag{D.35}$$

Now suppose $\gamma_t = \eta$ and $\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) \leq 0$. We can use $\gamma_t \leq \frac{\ell_{z_t}(\boldsymbol{w}_t)-\underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta}$ to write:

$$\begin{aligned}
\|\boldsymbol{w}_{t+1} - \mathbf{w}_\star\|^2 &\leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \gamma_t(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}}), \\
&\leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \frac{\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta}(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) \\
&\quad + \frac{\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta}(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}}),
\end{aligned} \tag{D.36}$$

where the last inequality has used $\gamma_t \leq \frac{\ell_{z_t}(\boldsymbol{w}_t)-\underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta}$, $\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) \leq 0$ and $\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}} \geq 0$. Therefore we are exactly in the same situation as the first case (where we used $\gamma_t = \frac{\ell_{z_t}(\boldsymbol{w}_t)-\underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta}$), and thus we have again:

$$\|\boldsymbol{w}_{t+1} - \mathbf{w}_\star\|^2 \leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \frac{(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star))^2}{C^2+\delta} + \frac{\varepsilon^2}{\delta}. \tag{D.37}$$

Now suppose that $\gamma_t = \eta$ and $\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) \geq 0$. The inequality (D.34) gives:

$$
\begin{aligned}
&\|\boldsymbol{w}_{t+1} - \mathbf{w}_\star\|^2 \\
&\quad \leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \gamma_t(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}}), \\
&\quad = \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \eta(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \gamma_t(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}}), \quad (\gamma_t = \eta) \\
&\quad \leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \eta(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \gamma_t\varepsilon, \quad (\text{definition of } \varepsilon,\ \gamma_t \geq 0) \\
&\quad \leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \eta(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \varepsilon\frac{\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}}}{\|\nabla\ell_{z_t}(\boldsymbol{w}_t)\|^2 + \delta}, \\
&\qquad (\text{because } \gamma_t \leq \frac{\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}}}{\|\nabla\ell_{z_t}(\boldsymbol{w}_t)\|^2 + \delta},\ \varepsilon \geq 0) \\
&\quad \leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \eta(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \varepsilon\frac{\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}}}{\delta}, \\
&\qquad (\text{because } \|\nabla\ell_{z_t}(\boldsymbol{w}_t)\|^2 \geq 0) \\
&\quad = \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \eta(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \varepsilon\frac{\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) + \ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}}}{\delta}, \\
&\quad \leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \eta(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \varepsilon\frac{\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) + \varepsilon}{\delta}, \\
&\qquad (\text{because } \ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}} \leq \varepsilon) \\
&\quad = \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \left(\eta - \frac{\varepsilon}{\delta}\right)(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \frac{\varepsilon^2}{\delta}.
\end{aligned}
\tag{D.38}
$$

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

$$
\begin{aligned}
\mathcal{I}_T &\triangleq \{t \in \{0, ..., T\} : \gamma_t = \eta \text{ and } \ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) \geq 0\} \\
\mathcal{J}_T &\triangleq \{0, ..., T\} \setminus \mathcal{I}_T
\end{aligned}
\tag{D.39}
$$

Then, by combining inequalities (D.35), (D.37) and (D.38), and using a telescopic sum, we obtain:

$$
\begin{aligned}
\|\boldsymbol{w}_{T+1} - \mathbf{w}_\star\|^2 \leq \|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 &+ \sum_{t\in\mathcal{J}_T}\left(-\frac{(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star))^2}{C^2 + \delta} + \frac{\varepsilon^2}{\delta}\right) \\
&+ \sum_{t\in\mathcal{I}_T}\left(-\left(\eta - \frac{\varepsilon}{\delta}\right)(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \frac{\varepsilon^2}{\delta}\right)
\end{aligned}
\tag{D.40}
$$

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

$$\begin{aligned}\frac{1}{C^2+\delta}\sum_{t\in\mathcal{J}_T}(\ell_{z_t}(\boldsymbol{w}_t)-\ell_{z_t}(\mathbf{w}_\star))^2 + \left(\eta-\frac{\varepsilon}{\delta}\right)\sum_{t\in\mathcal{I}_T}(\ell_{z_t}(\boldsymbol{w}_t)-\ell_{z_t}(\mathbf{w}_\star)) \\ \leq \|\boldsymbol{w}_0-\mathbf{w}_\star\|^2 + (T+1)\frac{\varepsilon^2}{\delta}\end{aligned} \tag{D.41}$$

In particular, the inequality (D.41) gives that:

$$\left(\eta-\frac{\varepsilon}{\delta}\right)\sum_{t\in\mathcal{I}_T}(\ell_{z_t}(\boldsymbol{w}_t)-\ell_{z_t}(\mathbf{w}_\star)) \leq \|\boldsymbol{w}_0-\mathbf{w}_\star\|^2 + (T+1)\frac{\varepsilon^2}{\delta}. \tag{D.42}$$

Furthermore, for every $t \in \mathcal{I}_T$, we have $(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) \geq 0$, which yields $\left(\eta - \frac{\varepsilon}{\delta}\right)\sum_{t\in\mathcal{I}_T}(\ell_{z_t}(\boldsymbol{w}_t)-\ell_{z_t}(\mathbf{w}_\star)) \geq 0$ since $\eta > \frac{\epsilon}{\delta}$. Thus the inequality (D.41) also gives:

$$\frac{1}{C^2+\delta}\sum_{t\in\mathcal{J}_T}(\ell_{z_t}(\boldsymbol{w}_t)-\ell_{z_t}(\mathbf{w}_\star))^2 \leq \|\boldsymbol{w}_0-\mathbf{w}_\star\|^2 + (T+1)\frac{\varepsilon^2}{\delta}. \tag{D.43}$$

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

$$\left(\sum_{t\in\mathcal{J}_T}\ell_{z_t}(\boldsymbol{w}_t)-\ell_{z_t}(\mathbf{w}_\star)\right)^2 \leq |\mathcal{J}_T|\sum_{t\in\mathcal{J}_T}(\ell_{z_t}(\boldsymbol{w}_t)-\ell_{z_t}(\mathbf{w}_\star))^2. \tag{D.44}$$

Therefore we have:

$$\begin{aligned}\sum_{t\in\mathcal{J}_T}\ell_{z_t}(\boldsymbol{w}_t)-\ell_{z_t}(\mathbf{w}_\star) &\leq \sqrt{|\mathcal{J}_T|\sum_{t\in\mathcal{J}_T}(\ell_{z_t}(\boldsymbol{w}_t)-\ell_{z_t}(\mathbf{w}_\star))^2}, \\ &\leq \sqrt{|\mathcal{J}_T|(C^2+\delta)\left(\|\boldsymbol{w}_0-\mathbf{w}_\star\|^2 + (T+1)\frac{\varepsilon^2}{\delta}\right)}.\end{aligned} \tag{D.45}$$

We can now put together inequalities (D.42) and (D.44) by writing:

$$\begin{aligned}
&\sum_{t=0}^{T} \ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) \\
&\quad = \sum_{t\in\mathcal{I}_T} \ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) + \sum_{t\in\mathcal{J}_T} \ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) \\
&\quad \leq \frac{1}{\eta - \frac{\varepsilon}{\delta}} \left(\|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 + (T+1)\frac{\varepsilon^2}{\delta} \right) \\
&\qquad + \sqrt{|\mathcal{J}_T|(C^2+\delta)\left(\|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 + (T+1)\frac{\varepsilon^2}{\delta} \right)} \\
&\quad \leq \frac{1}{\eta - \frac{\varepsilon}{\delta}} \left(\|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 + (T+1)\frac{\varepsilon^2}{\delta} \right) \\
&\qquad + \sqrt{(T+1)(C^2+\delta)\left(\|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 + (T+1)\frac{\varepsilon^2}{\delta} \right)}
\end{aligned} \tag{D.46}$$

Dividing by $T+1$ and taking the expectation, we obtain:

$$\begin{aligned}
&f\left(\frac{1}{T+1}\sum_{t=0}^{T} \boldsymbol{w}_t \right) - f_\star \\
&\quad \leq \frac{1}{T+1}\sum_{t=0}^{T} f(\boldsymbol{w}_t) - f_\star, \quad (f \text{ is convex}) \\
&\quad \leq \frac{\|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2}{(\eta - \frac{\varepsilon}{\delta})(T+1)} + \frac{\varepsilon^2}{\delta(\eta - \frac{\varepsilon}{\delta})} + \sqrt{(C^2+\delta)\left(\frac{\|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2}{T+1} + \frac{\varepsilon^2}{\delta} \right)}, \\
&\quad \leq \frac{\|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2}{(\eta - \frac{\varepsilon}{\delta})(T+1)} + \frac{\varepsilon^2}{\delta(\eta - \frac{\varepsilon}{\delta})} + \sqrt{\frac{(C^2+\delta)\|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2}{T+1}} + \varepsilon\sqrt{\frac{C^2}{\delta} + 1}.
\end{aligned} \tag{D.47}$$

□

D.4.5 Theorem 5

Theorem 5. *We assume that $\mathcal{X}$ is a convex set, and that for every $z \in \mathcal{Z}$, ℓ_z is convex and C-Lipschitz. Let $\mathbf{w}_\star$ be an ε-interpolation for ($(\mathcal{P})$, B). Then if we apply ALI-G with a maximal learning-rate of η to f, we have:*

$$f\left(\frac{1}{T+1}\sum_{t=0}^{T} \boldsymbol{w}_t \right) - f_\star \leq \frac{\|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2}{\eta(T+1)} + \varepsilon + \sqrt{\frac{(C^2+\delta)\|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2}{T+1}} + \eta\varepsilon\sqrt{C^2+\delta}. \tag{D.7}$$

Proof.

We consider the update at time t, which we condition on the draw of $z_t \in \mathcal{Z}$. We re-use the inequality (D.34) from the proof of Theorem 4:

$$\|\boldsymbol{w}_{t+1} - \mathbf{w}_\star\|^2 \leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \gamma_t(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}}) \quad \text{(D.48)}$$

We consider again different cases, according to the value of γ_t and the sign of $\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)$.

Suppose that $\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) \leq 0$. Then the inequality (D.48) gives:

$$\begin{aligned}
&\|\boldsymbol{w}_{t+1} - \mathbf{w}_\star\|^2 \\
&\quad \leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \gamma_t(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}}), \\
&\quad \leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \eta(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \gamma_t(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}}), \\
&\qquad \text{(because } \gamma_t \leq \eta,\ \ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) \leq 0) \\
&\quad \leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \eta(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \gamma_t\varepsilon, \quad \text{(definition of } \varepsilon,\ \gamma_t \geq 0) \\
&\quad \leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \eta(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \eta\varepsilon, \quad (\gamma_t \leq \eta,\ \varepsilon \geq 0)
\end{aligned} \quad \text{(D.49)}$$

Now suppose $\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) \geq 0$ and $\gamma_t = \eta$. Then the inequality (D.48) gives:

$$\begin{aligned}
\|\boldsymbol{w}_{t+1} - \mathbf{w}_\star\|^2 &\leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \gamma_t(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}}), \\
&= \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \eta(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \eta(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}}), \\
&\qquad \text{(because } \gamma_t = \eta) \\
&\leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \eta(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \eta\varepsilon, \\
&\qquad \text{(definition of } \varepsilon,\ \eta \geq 0)
\end{aligned} \quad \text{(D.50)}$$

Finally, suppose that $\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) \geq 0$ and $\gamma_t = \frac{\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 + \delta}$. Then the inequality (D.48) gives:

$$
\begin{aligned}
&\|\boldsymbol{w}_{t+1} - \mathbf{w}_\star\|^2 \\
&\quad \leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \gamma_t(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}}), \\
&\quad \leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \eta(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}}), \\
&\qquad \text{(because } \gamma_t \leq \eta,\ \ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}} \geq 0) \\
&\quad \leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \eta\varepsilon, \quad \text{(definition of } \varepsilon,\ \eta \geq 0) \\
&\quad = \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \frac{\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 + \delta}(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \eta\varepsilon, \\
&\qquad \text{(because } \gamma_t = \frac{\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 + \delta}) \\
&\quad \leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \frac{(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star))^2}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 + \delta} + \eta\varepsilon, \\
&\qquad \text{(because } \ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}} \geq \ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) \geq 0) \\
&\quad \leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \frac{(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star))^2}{C^2 + \delta} + \eta\varepsilon, \quad (\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 \leq C^2)
\end{aligned}
\tag{D.51}
$$

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

$$
\begin{aligned}
\mathcal{J}_T &\triangleq \left\{ t \in \{0, ..., T\} : \gamma_t = \frac{\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 + \delta} \text{ and } \ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) \geq 0 \right\} \\
\mathcal{I}_T &\triangleq \{0, ..., T\} \setminus \mathcal{I}_T
\end{aligned}
\tag{D.52}
$$

Then, by combining inequalities (D.49), (D.50) and (D.51), and using a telescopic sum, we obtain:

$$
\begin{aligned}
\|\boldsymbol{w}_{T+1} - \mathbf{w}_\star\|^2 \leq \|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 &+ \sum_{t \in \mathcal{J}_T} \left(-\frac{(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star))^2}{C^2 + \delta} + \eta\varepsilon \right) \\
&+ \sum_{t \in \mathcal{I}_T} (-\eta(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \eta\varepsilon)
\end{aligned}
\tag{D.53}
$$

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

$$
\begin{aligned}
&\frac{1}{C^2 + \delta} \sum_{t \in \mathcal{J}_T} (\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star))^2 + \eta \sum_{t \in \mathcal{I}_T} (\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) \\
&\quad \leq \|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 + (T+1)\eta\varepsilon
\end{aligned}
\tag{D.54}
$$

We now take the expectation and obtain:

$$\begin{aligned}&\frac{1}{C^2+\delta}\sum_{t\in\mathcal{J}_T}\mathbb{E}\left[(\ell_{z_t}(\boldsymbol{w}_t)-\ell_{z_t}(\mathbf{w}_\star))^2\right]+\eta\sum_{t\in\mathcal{I}_T}(f(\boldsymbol{w}_t)-f_\star)\\&\quad\leq\|\boldsymbol{w}_0-\mathbf{w}_\star\|^2+(T+1)\eta\varepsilon\end{aligned}\tag{D.55}$$

Since $\mathbb{E}[U]^2\leq\mathbb{E}[U^2]$ for any real-valued random variable, we can write:

$$\frac{1}{C^2+\delta}\sum_{t\in\mathcal{J}_T}(f(\boldsymbol{w}_t)-f_\star)^2+\eta\sum_{t\in\mathcal{I}_T}(f(\boldsymbol{w}_t)-f_\star)\leq\|\boldsymbol{w}_0-\mathbf{w}_\star\|^2+(T+1)\eta\varepsilon\tag{D.56}$$

Since each $f(\boldsymbol{w}_t)-f_\star\geq 0$, the inequality (D.56) gives that:

$$\eta\sum_{t\in\mathcal{I}_T}(f(\boldsymbol{w}_t)-f_\star)\leq\|\boldsymbol{w}_0-\mathbf{w}_\star\|^2+(T+1)\eta\varepsilon,\tag{D.57}$$

and:

$$\frac{1}{C^2+\delta}\sum_{t\in\mathcal{J}_T}(f(\boldsymbol{w}_t)-f_\star)^2\leq\|\boldsymbol{w}_0-\mathbf{w}_\star\|^2+(T+1)\eta\varepsilon.\tag{D.58}$$

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

$$\left(\sum_{t\in\mathcal{J}_T}f(\boldsymbol{w}_t)-f_\star\right)^2\leq|\mathcal{J}_T|\sum_{t\in\mathcal{J}_T}(f(\boldsymbol{w}_t)-f_\star)^2.\tag{D.59}$$

Therefore we have:

$$\begin{aligned}\sum_{t\in\mathcal{J}_T}f(\boldsymbol{w}_t)-f_\star&\leq\sqrt{|\mathcal{J}_T|\sum_{t\in\mathcal{J}_T}(f(\boldsymbol{w}_t)-f_\star)^2},\\&\leq\sqrt{|\mathcal{J}_T|(C^2+\delta)\left(\|\boldsymbol{w}_0-\mathbf{w}_\star\|^2+(T+1)\eta\varepsilon\right)}.\end{aligned}\tag{D.60}$$

We can now put together inequalities (D.57) and (D.60) by writing:

$$\begin{aligned}&\sum_{t=0}^{T}f(\boldsymbol{w}_t)-f_\star\\&\quad=\sum_{t\in\mathcal{I}_T}f(\boldsymbol{w}_t)-f_\star+\sum_{t\in\mathcal{J}_T}f(\boldsymbol{w}_t)-f_\star\\&\quad\leq\frac{1}{\eta}\left(\|\boldsymbol{w}_0-\mathbf{w}_\star\|^2+(T+1)\eta\varepsilon\right)+\sqrt{|\mathcal{J}_T|(C^2+\delta)\left(\|\boldsymbol{w}_0-\mathbf{w}_\star\|^2+(T+1)\eta\varepsilon\right)}\\&\quad\leq\frac{1}{\eta}\left(\|\boldsymbol{w}_0-\mathbf{w}_\star\|^2+(T+1)\eta\varepsilon\right)\\&\qquad+\sqrt{(T+1)(C^2+\delta)\left(\|\boldsymbol{w}_0-\mathbf{w}_\star\|^2+(T+1)\eta\varepsilon\right)}\end{aligned}\tag{D.61}$$

Dividing by $T+1$, we obtain:

$$\begin{aligned} & f\left(\frac{1}{T+1}\sum_{t=0}^{T}\boldsymbol{w}_t\right) - f_\star \\ & \leq \frac{1}{T+1}\sum_{t=0}^{T} f(\boldsymbol{w}_t) - f_\star, \quad (f \text{ is convex}) \\ & \leq \frac{\|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2}{\eta(T+1)} + \varepsilon + \sqrt{(C^2+\delta)\left(\frac{\|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2}{T+1} + \eta\varepsilon\right)}, \\ & \leq \frac{\|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2}{\eta(T+1)} + \varepsilon + \sqrt{\frac{(C^2+\delta)\|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2}{T+1}} + \eta\varepsilon\sqrt{C^2+\delta}. \end{aligned} \tag{D.62}$$

□

D.4.6 Theorem 6

Lemma 10. *Let $z \in \mathcal{Z}$. Assume that ℓ_z is convex, β-smooth and is lower-bounded on $\mathbb{R}^p$ by $\underline{B} \in \mathbb{R}$. Then we have:*

$$\forall\, \boldsymbol{w} \in \mathbb{R}^p,\ \ell_z(\boldsymbol{w}) - \underline{B} \geq \frac{1}{2\beta}\|\nabla\ell_z(\boldsymbol{w})\|^2 \tag{D.63}$$

Proof. Let $\boldsymbol{w} \in \mathbb{R}^p$ and suppose that ℓ_z reaches its infimum at $\underline{\mathbf{w}} \in (\mathbb{R} \cup \{-\infty, +\infty\})^p$.

First, let us consider the case $\underline{\mathbf{w}} \in \mathbb{R}^p$. Then by Lemma 3.5 of (Bubeck, 2015), we have:

$$\begin{aligned} \ell_z(\underline{\mathbf{w}}) - \ell_z(\boldsymbol{w}) &\leq \nabla\ell_z(\underline{\mathbf{w}})^\top(\underline{\mathbf{w}} - \boldsymbol{w}) - \frac{1}{2\beta}\|\nabla\ell_z(\underline{\mathbf{w}}) - \nabla\ell_z(\boldsymbol{w})\|^2, \\ &= -\frac{1}{2\beta}\|\nabla\ell_z(\boldsymbol{w})\|^2 \quad (\nabla\ell_z(\underline{\mathbf{w}}) = \mathbf{0}). \end{aligned} \tag{D.64}$$

Therefore we can write:

$$\ell_z(\boldsymbol{w}) - \underline{\mathrm{B}} \geq \ell_z(\boldsymbol{w}) - \ell_z(\underline{\mathbf{w}}) \geq \frac{1}{2\beta}\|\nabla\ell_z(\boldsymbol{w})\|^2. \tag{D.65}$$

Now let us assume that $\underline{\mathbf{w}} \notin \mathbb{R}^p$. Then we can construct a sequence $(\underline{\mathbf{w}}_k)_{k\in\mathbb{N}} \in (\mathbb{R}^p)^{\mathbb{N}}$ that converges to $\underline{\mathbf{w}}$. Since ℓ_z and $\nabla\ell_z$ are continuous functions (they are respectively convex and smooth), we have:

$$\begin{aligned} \lim_{k\to\infty} \ell_z(\underline{\mathbf{w}}_k) &= \inf \ell_z, \\ \lim_{k\to\infty} \nabla\ell_z(\underline{\mathbf{w}}_k) &= \mathbf{0}. \end{aligned} \tag{D.66}$$

Therefore the previous case gives the wanted result by using $\underline{\mathbf{w}}_k$ in place of $\underline{\mathbf{w}}$ and then taking the limit $k \to \infty$.

□

Lemma 11. *Let $z \in \mathcal{Z}$. Assume that ℓ_z is convex, β-smooth and is lower-bounded on $\mathbb{R}^p$ by $\underline{B} \in \mathbb{R}$. Then we have:*

$$\forall\, \boldsymbol{w} \in \mathbb{R}^p,\ \frac{\ell_z(\boldsymbol{w}) - \underline{B}}{\|\nabla \ell_z(\boldsymbol{w})\|^2 + \delta} \geq \frac{1}{2\beta} - \frac{\delta}{4\beta^2(\ell_z(\boldsymbol{w}) - \underline{B}))} \tag{D.67}$$

Proof.

Let $\boldsymbol{w} \in \mathbb{R}^p$. We apply Lemma 10 and we write successively:

$$\begin{aligned}
\frac{\ell_z(\boldsymbol{w}) - \underline{\mathrm{B}}}{\|\nabla \ell_z(\boldsymbol{w})\|^2 + \delta} &\geq \frac{\ell_z(\boldsymbol{w}) - \underline{\mathrm{B}}}{2\beta(\ell_z(\boldsymbol{w}) - \underline{\mathrm{B}}) + \delta}, \quad \text{(Lemma 10)} \\
&= \frac{\ell_z(\boldsymbol{w}) - \underline{\mathrm{B}} + \frac{\delta}{2\beta} - \frac{\delta}{2\beta}}{2\beta(\ell_z(\boldsymbol{w}) - \underline{\mathrm{B}} + \frac{\delta}{2\beta})}, \\
&= \frac{1}{2\beta} - \frac{\frac{\delta}{2\beta}}{2\beta(\ell_z(\boldsymbol{w}) - \underline{\mathrm{B}} + \frac{\delta}{2\beta})}, \\
&\geq \frac{1}{2\beta} - \frac{\delta}{4\beta^2(\ell_z(\boldsymbol{w}) - \underline{\mathrm{B}})}. \quad (\delta \geq 0)
\end{aligned} \tag{D.68}$$

□

Theorem 6. *[Convex and Smooth] We assume that $\mathcal{X}$ is a convex set, and that for every $z \in \mathcal{Z}$, ℓ_z is convex and β-smooth. Let $\underline{B}$ be a uniform lower bound on $(\mathcal{P})$ and $\mathbf{w}_\star$ be a solution of $(\mathcal{P})$. Further suppose that $\mathbf{w}_\star$ is an ε-interpolation for $((\mathcal{P}), \underline{B})$, and that $\delta > 2\beta\varepsilon$. Then ALI-G$^\infty$ applied to f satisfies:*

$$f\left(\frac{1}{T+1}\sum_{t=0}^{T} \boldsymbol{w}_t\right) - f_\star \leq \frac{\delta}{\beta(1 - \frac{2\beta\varepsilon}{\delta})} + \frac{2\beta}{1 - \frac{2\beta\varepsilon}{\delta}} \frac{\|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2}{T+1}. \tag{D.8}$$

Proof.

We consider the update at time t, which we condition on the draw of $z_t \in \mathcal{Z}$:

$$\begin{aligned}
&\|\boldsymbol{w}_{t+1} - \mathbf{w}_\star\|^2 \\
&\quad = \|\Pi_\Omega(\boldsymbol{w}_t - \gamma_t \nabla \ell_{z_t}(\boldsymbol{w}_t)) - \mathbf{w}_\star\|^2 \\
&\quad \leq \|\boldsymbol{w}_t - \gamma_t \nabla \ell_{z_t}(\boldsymbol{w}_t) - \mathbf{w}_\star\|^2 \quad (\Pi_\Omega \text{ projection}) \\
&\quad = \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - 2\gamma_t \nabla \ell_{z_t}(\boldsymbol{w}_t)^\top (\boldsymbol{w}_t - \mathbf{w}_\star) + \gamma_t^2 \|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 \\
&\quad = \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - 2\gamma_t \nabla \ell_{z_t}(\boldsymbol{w}_t)^\top (\boldsymbol{w}_t - \mathbf{w}_\star) + \gamma_t \frac{\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 + \delta} \|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2
\end{aligned}$$

$$
\begin{aligned}
&\quad \text{(definition of } \gamma_t\text{)} \\
&\leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - 2\gamma_t \nabla \ell_{z_t}(\boldsymbol{w}_t)^\top (\boldsymbol{w}_t - \mathbf{w}_\star) + \gamma_t \frac{\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2} \|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 \\
&\quad \text{(because } \ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}} \geq 0 \text{ and } \delta \geq 0\text{)} \\
&\leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - 2\gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}}) \quad \text{(convexity of } \ell_{z_t}\text{)} \\
&= \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - 2\gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) \\
&\quad + \gamma_t(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}}) \\
&= \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \gamma_t(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}}) \qquad \text{(D.69)}
\end{aligned}
$$

We now lower bound $\gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star))$ and upper bound $\gamma_t(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}})$ individually.

We begin with $\gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star))$, for which we distinguish two cases according to its sign:

Suppose $(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) \geq 0$. Then we can write:

$$
\begin{aligned}
&\gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) \\
&\quad = \frac{\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 + \delta} (\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)), \quad \text{(definition of } \gamma_t\text{)} \\
&\quad \geq \left(\frac{1}{2\beta} - \frac{\delta}{4\beta^2(\ell_z(\boldsymbol{w}_t) - \underline{\mathrm{B}})} \right) (\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) \\
&\qquad \text{(using Lemma 11, } (\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) \geq 0\text{)} \\
&\quad = \frac{1}{2\beta}(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) - \frac{\delta}{4\beta^2} \frac{\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)}{(\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}})} \\
&\quad \geq \frac{1}{2\beta}(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) - \frac{\delta}{4\beta^2} \quad (\ell_{z_t}(\mathbf{w}_\star) \geq \underline{\mathrm{B}},\ (\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) \geq 0) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(D.70)}
\end{aligned}
$$

Now suppose $(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) \leq 0$, and let us show that the same result holds. We have:

$$\begin{aligned}
\gamma_t &= \frac{\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 + \delta} \\
&\leq \frac{\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 + \delta} \quad ((\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) \leq 0) \\
&\leq \frac{\varepsilon}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 + \delta} \quad \text{(definition of } \varepsilon) \\
&\leq \frac{\varepsilon}{\delta} \quad (\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\| \geq 0) \\
&\leq \frac{1}{2\beta} \quad (\delta \geq 2\beta\varepsilon)
\end{aligned} \tag{D.71}$$

Therefore we have:

$$\begin{aligned}
&\gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) \\
&\quad \geq \frac{1}{2\beta}(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) \quad ((\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) \leq 0) \\
&\quad \geq \frac{1}{2\beta}(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) - \frac{\delta}{4\beta^2} \quad (\delta \geq 0)
\end{aligned} \tag{D.72}$$

In conclusion, regardless of the sign, it always holds true that:

$$\gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) \geq \frac{1}{2\beta}(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) - \frac{\delta}{4\beta^2} \tag{D.73}$$

We now upper bound $\gamma_t(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}})$:

$$\begin{aligned}
\gamma_t(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}}) &= \frac{(\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}})(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}})}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 + \delta}, \quad \text{(definition of } \gamma_t) \\
&\leq \frac{(\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}})(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}})}{\delta}, \quad (\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\| \geq 0) \\
&\leq \frac{(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) + \varepsilon)\varepsilon}{\delta}, \quad \text{(definition of } \varepsilon \text{ twice)} \\
&= \frac{\varepsilon}{\delta}((\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \frac{\varepsilon^2}{\delta}.
\end{aligned} \tag{D.74}$$

Putting inequalities (D.69), (D.73) and (D.74) together, we obtain:

$$\begin{aligned}
&\|\boldsymbol{w}_{t+1} - \mathbf{w}_\star\|^2 \\
&\quad \leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \frac{1}{2\beta}(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \frac{\delta}{4\beta^2} + \frac{\varepsilon}{\delta}((\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \frac{\varepsilon^2}{\delta}.
\end{aligned} \tag{D.75}$$

Therefore we have:

$$\left(\frac{1}{2\beta}-\frac{\varepsilon}{\delta}\right)\left(\ell_{z_t}(\boldsymbol{w}_t)-\ell_{z_t}(\mathbf{w}_\star)\right)-\left(\frac{\delta}{4\beta^2}+\frac{\varepsilon^2}{\delta}\right)\leq\|\boldsymbol{w}_t-\mathbf{w}_\star\|^2-\|\boldsymbol{w}_{t+1}-\mathbf{w}_\star\|^2. \tag{D.76}$$

By summing over t and taking the expectation over the z_t, we obtain:

$$\begin{aligned}&\frac{\delta-2\beta\varepsilon}{2\beta\delta}\sum_{t=0}^{T}\left(f(\boldsymbol{w}_t)-f(\mathbf{w}_\star)-\frac{\delta^2+4\beta^2\varepsilon^2}{4\beta^2\delta}\right)\\&\quad\leq\|\boldsymbol{w}_0-\mathbf{w}_\star\|^2-\mathbb{E}\left[\|\boldsymbol{w}_{T+1}-\mathbf{w}_\star\|^2\right],\\&\quad\leq\|\boldsymbol{w}_0-\mathbf{w}_\star\|^2.\end{aligned} \tag{D.77}$$

By assumption, we have that $\delta-2\beta\varepsilon>0$. Dividing by $T+1$ and using the convexity of f, we finally obtain:

$$\begin{aligned}f\left(\frac{1}{T+1}\sum_{t=0}^{T}\boldsymbol{w}_t\right)-f_\star&\leq\frac{1}{T+1}\sum_{t=0}^{T}f(\boldsymbol{w}_t)-f_\star\quad\text{(convexity of }f\text{)},\\&=\frac{2\beta\delta}{\delta-2\beta\varepsilon}\frac{\delta^2+4\beta^2\varepsilon^2}{4\beta^2\delta}+\frac{2\beta\delta}{\delta-2\beta\varepsilon}\frac{\|\boldsymbol{w}_0-\mathbf{w}_\star\|^2}{T+1},\\&=\frac{\delta^2+4\beta^2\varepsilon^2}{2\beta(\delta-2\beta\varepsilon)}+\frac{2\beta\delta}{\delta-2\beta\varepsilon}\frac{\|\boldsymbol{w}_0-\mathbf{w}_\star\|^2}{T+1},\\&\leq\frac{\delta^2}{\beta(\delta-2\beta\varepsilon)}+\frac{2\beta\delta}{\delta-2\beta\varepsilon}\frac{\|\boldsymbol{w}_0-\mathbf{w}_\star\|^2}{T+1},\quad(\delta-2\beta\varepsilon\geq0)\\&=\frac{\delta}{\beta(1-\frac{2\beta\varepsilon}{\delta})}+\frac{2\beta}{1-\frac{2\beta\varepsilon}{\delta}}\frac{\|\boldsymbol{w}_0-\mathbf{w}_\star\|^2}{T+1}.\end{aligned} \tag{D.78}$$

□

D.4.7 Theorem 7

Theorem 7. *We assume that $\mathcal{X}$ is a convex set, and that for every $z\in\mathcal{Z}$, ℓ_z is convex and β-smooth. Let $\mathbf{w}_\star$ be an ε-interpolation for $((\mathcal{P}),\underline{B})$, and suppose that $\delta>2\beta\varepsilon$. Further assume that $\eta\geq\frac{1}{2\beta}$. Then if we apply ALI-G with a maximal learning-rate of η to f, we have:*

$$f\left(\frac{1}{T+1}\sum_{t=0}^{T}\boldsymbol{w}_t\right)-f_\star\leq\frac{\delta}{\beta(1-\frac{2\beta\varepsilon}{\delta})}+\frac{2\beta}{1-\frac{2\beta\varepsilon}{\delta}}\frac{\|\boldsymbol{w}_0-\mathbf{w}_\star\|^2}{T+1}. \tag{D.9}$$

Proof.

By using the fact that $\gamma_t \leq \frac{\ell_{z_t}(\boldsymbol{w}_t)-\underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta}$ rather than $\gamma_t = \frac{\ell_{z_t}(\boldsymbol{w}_t)-\underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta}$, we can see that the inequality (D.69) is still valid:

$$\|\boldsymbol{w}_{t+1} - \mathbf{w}_\star\|^2 \leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \gamma_t(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}}) \quad \text{(D.79)}$$

As previously, we lower bound $\gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star))$ and upper bound $\gamma_t(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}})$ individually.

We begin with $\gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star))$. We remark that either $\gamma_t = \frac{\ell_{z_t}(\boldsymbol{w}_t)-\underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta}$ or $\gamma_t = \eta$.

Suppose $\gamma_t = \frac{\ell_{z_t}(\boldsymbol{w}_t)-\underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta}$ and $\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) \geq 0$. Then we are in the same condition as in Theorem 6, and thus the inequality (D.73) holds true:

$$\gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) \geq \frac{1}{2\beta}(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) - \frac{\delta}{4\beta^2}. \quad \text{(D.80)}$$

Now suppose $\gamma_t = \eta$ and $\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) \geq 0$. Then we have:

$$\begin{aligned}
\gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) &= \eta(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) \\
&\geq \eta(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) - \frac{\delta}{4\beta^2} \\
&\geq \frac{1}{2\beta}(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) - \frac{\delta}{4\beta^2} \\
&\qquad \text{(because } \eta \geq \frac{1}{2\beta},\ \ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) \geq 0\text{).}
\end{aligned} \quad \text{(D.81)}$$

Now suppose $\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) \leq 0$. By using $\gamma_t \leq \frac{\ell_{z_t}(\boldsymbol{w}_t)-\underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta}$ instead of $\gamma_t = \frac{\ell_{z_t}(\boldsymbol{w}_t)-\underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta}$, we can see that the inequality (D.71) is still valid, which gives:

$$\gamma_t \leq \frac{1}{2\beta} \quad \text{(D.82)}$$

We now use $\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) \leq 0$ to write:

$$\begin{aligned}
\gamma_t\left(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)\right) &\geq \frac{1}{2\beta}(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) \quad (\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) \leq 0) \\
&\geq \frac{1}{2\beta}(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) - \frac{\delta}{4\beta^2}
\end{aligned} \quad \text{(D.83)}$$

In conclusion, in all cases, it holds true that:

$$\gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) \geq \frac{1}{2\beta}(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) - \frac{\delta}{4\beta^2} \tag{D.84}$$

By using $\gamma_t \leq \frac{\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 + \delta}$, we can remark that the inequality (D.74) holds true and gives:

$$\gamma_t(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}}) \leq \frac{\varepsilon}{\delta}((\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \frac{\varepsilon^2}{\delta}. \tag{D.85}$$

We now put together inequalities (D.79), (D.84) and (D.85):

$$\begin{aligned}
&\|\boldsymbol{w}_{t+1} - \mathbf{w}_\star\|^2 \\
&\quad \leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \frac{1}{2\beta}(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \frac{\delta}{4\beta^2} + \frac{\varepsilon}{\delta}((\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \frac{\varepsilon^2}{\delta}, \\
&\quad = \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \left(\frac{1}{2\beta} - \frac{\varepsilon}{\delta}\right)(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \frac{\delta}{4\beta^2} + \frac{\varepsilon^2}{\delta}.
\end{aligned} \tag{D.86}$$

This is exactly the same result as in the inequality (D.76) from the proof of Theorem 6. Therefore the rest of the proof of Theorem 6 follows and we obtain the desired result.

□

D.4.8 Theorem 8

Theorem 8. *We assume that $\mathcal{X}$ is a convex set, and that for every $z \in \mathcal{Z}$, ℓ_z is convex and β-smooth. Let $\mathbf{w}_\star$ be an ε-interpolation for $((\mathcal{P}), \underline{\mathrm{B}})$, and suppose that $\delta > 2\beta\varepsilon$. Further assume that $\eta \leq \frac{1}{2\beta}$. Then if we apply ALI-G with a maximal learning-rate of η to f, we have:*

$$f\left(\frac{1}{T+1}\sum_{t=0}^{T}\boldsymbol{w}_t\right) - f_\star \leq \frac{\|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2}{\eta(T+1)} + \frac{\delta}{2\beta} + \varepsilon. \tag{D.10}$$

Proof.

By using the fact that $\gamma_t \leq \frac{\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 + \delta}$ rather than $\gamma_t = \frac{\ell_{z_t}(\boldsymbol{w}_t) - \underline{\mathrm{B}}}{\|\nabla \ell_{z_t}(\boldsymbol{w}_t)\|^2 + \delta}$, we can see that the inequality (D.69) is still valid:

$$\|\boldsymbol{w}_{t+1} - \mathbf{w}_\star\|^2 \leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \gamma_t(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}}) \tag{D.87}$$

As previously, we lower bound $\gamma_t(\ell_{z_t}(\boldsymbol{w}_t)-\ell_{z_t}(\mathbf{w}_\star))$ and upper bound $\gamma_t(\ell_{z_t}(\mathbf{w}_\star)-\underline{\mathrm{B}})$ individually.

We begin with $\gamma_t(\ell_{z_t}(\boldsymbol{w}_t)-\ell_{z_t}(\mathbf{w}_\star))$. We remark that either $\gamma_t=\frac{\ell_{z_t}(\boldsymbol{w}_t)-\underline{\mathrm{B}}}{\|\nabla\ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta}$ or $\gamma_t=\eta$.

Suppose $\gamma_t=\frac{\ell_{z_t}(\boldsymbol{w}_t)-\underline{\mathrm{B}}}{\|\nabla\ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta}$ and $\ell_{z_t}(\boldsymbol{w}_t)-\ell_{z_t}(\mathbf{w}_\star)\geq 0$. First we write:

$$\begin{aligned}
\gamma_t &= \frac{\ell_{z_t}(\boldsymbol{w}_t)-\underline{\mathrm{B}}}{\|\ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta}\\
&= \frac{\ell_{z_t}(\boldsymbol{w}_t)-\underline{\mathrm{B}}+\frac{\delta}{2\beta}}{\|\ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta}-\frac{\frac{\delta}{2\beta}}{\|\ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta}\\
&\geq \frac{\frac{\|\ell_{z_t}(\boldsymbol{w}_t)\|^2}{2\beta}+\frac{\delta}{2\beta}}{\|\ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta}-\frac{\delta}{2\beta}\frac{1}{\|\ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta} \quad \text{(Lemma 10)}\\
&= \frac{1}{2\beta}-\frac{\delta}{2\beta}\frac{1}{\|\ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta}\\
&\geq \eta-\frac{\delta}{2\beta}\frac{1}{\|\ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta} \quad (\eta\leq\frac{1}{2\beta})
\end{aligned} \tag{D.88}$$

Since $\ell_{z_t}(\boldsymbol{w}_t)-\ell_{z_t}(\mathbf{w}_\star)\geq 0$, this yields:

$$\begin{aligned}
\gamma_t(\ell_{z_t}(\boldsymbol{w}_t)-\ell_{z_t}(\mathbf{w}_\star)) &\geq \left(\eta-\frac{\delta}{2\beta}\frac{1}{\|\ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta}\right)(\ell_{z_t}(\boldsymbol{w}_t)-\ell_{z_t}(\mathbf{w}_\star))\\
&= \eta(\ell_{z_t}(\boldsymbol{w}_t)-\ell_{z_t}(\mathbf{w}_\star))-\frac{\delta}{2\beta}\frac{\ell_{z_t}(\boldsymbol{w}_t)-\ell_{z_t}(\mathbf{w}_\star)}{\|\ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta}\\
&\geq \eta(\ell_{z_t}(\boldsymbol{w}_t)-\ell_{z_t}(\mathbf{w}_\star))-\frac{\delta}{2\beta}\frac{\ell_{z_t}(\boldsymbol{w}_t)-\underline{\mathrm{B}}}{\|\ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta}\\
&\qquad \text{(because } \ell_{z_t}(\mathbf{w}_\star)\geq\underline{\mathrm{B}})
\end{aligned} \tag{D.89}$$

We now notice that since $\gamma_t=\frac{\ell_{z_t}(\boldsymbol{w}_t)-\underline{\mathrm{B}}}{\|\nabla\ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta}$, and $\gamma_t\leq\eta$, then necessarily $\frac{\ell_{z_t}(\boldsymbol{w}_t)-\underline{\mathrm{B}}}{\|\nabla\ell_{z_t}(\boldsymbol{w}_t)\|^2+\delta}\leq\eta$. This gives:

$$\gamma_t(\ell_{z_t}(\boldsymbol{w}_t)-\ell_{z_t}(\mathbf{w}_\star))\geq\eta(\ell_{z_t}(\boldsymbol{w}_t)-\ell_{z_t}(\mathbf{w}_\star))-\frac{\eta\delta}{2\beta} \tag{D.90}$$

Now suppose $\gamma_t = \eta$ and $\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) \geq 0$. Then we have:

$$\begin{aligned}\gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) &= \eta(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) \\ &\geq \eta(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) - \frac{\eta\delta}{2\beta}.\end{aligned} \tag{D.91}$$

Now suppose $\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) \leq 0$. Since $\gamma_t \leq \eta$ by definition, we have that:

$$\begin{aligned}\gamma_t\left(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)\right) &\geq \eta(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) \quad (\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) \leq 0) \\ &\geq \eta(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) - \frac{\eta\delta}{2\beta}.\end{aligned} \tag{D.92}$$

In conclusion, in all cases, it holds true that:

$$\gamma_t(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) \geq \eta(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) - \frac{\eta\delta}{2\beta} \tag{D.93}$$

We upper bound $\gamma_t(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}})$ as follows:

$$\begin{aligned}\gamma_t(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}}) &\leq \eta(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}}) \quad (\ell_{z_t}(\mathbf{w}_\star) \geq \underline{\mathrm{B}}) \\ &\leq \eta\varepsilon \quad (\text{definition of } \varepsilon)\end{aligned} \tag{D.94}$$

We combine inequalities (D.87), (D.93) and (D.94) and obtain:

$$\|\boldsymbol{w}_{t+1} - \mathbf{w}_\star\|^2 \leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \eta(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \frac{\eta\delta}{2\beta} + \eta\varepsilon. \tag{D.95}$$

By taking the expectation and using a telescopic sum, we obtain:

$$0 \leq \|\boldsymbol{w}_{T+1} - \mathbf{w}_\star\|^2 \leq \|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 - \sum_{t=0}^{T}\left(\eta(f(\boldsymbol{w}_t) - f_\star) + \frac{\eta\delta}{2\beta} + \eta\varepsilon\right). \tag{D.96}$$

Re-arranging and using the convexity of f, we finally obtain:

$$f\left(\frac{1}{T+1}\sum_{t=0}^{T}\boldsymbol{w}_t\right) \leq \frac{\|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2}{\eta(T+1)} + \frac{\delta}{2\beta} + \varepsilon. \tag{D.97}$$

□

D.4.9 Theorem 9

Lemma 12. *For any $a, b \in \mathbb{R}^p$, we have that:*

$$\|a\|^2 + \|b\|^2 \geq \frac{1}{2}\|a - b\|^2 \tag{D.98}$$

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 13. *Let $z \in \mathcal{Z}$. Assume that ℓ_z is α-strongly convex and is lower-bounded on $\mathbb{R}^p$ by $\underline{B} \in \mathbb{R}$ such that $\inf \ell_z - \underline{B} \leq \varepsilon$. In addition, suppose that $\delta \geq 2\alpha\varepsilon$. Then we have:*

$$\forall\, \boldsymbol{w} \in \mathbb{R}^p,\ \frac{\ell_z(\boldsymbol{w}) - \underline{B}}{\|\nabla \ell_z(\boldsymbol{w})\|^2 + \delta} \leq \frac{1}{2\alpha}. \tag{D.99}$$

Proof.

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

$$\forall\, \hat{\boldsymbol{w}} \in \mathbb{R}^p,\ \ell_z(\hat{\boldsymbol{w}}) \geq \ell_z(\boldsymbol{w}) + \nabla \ell_z(\boldsymbol{w})^\top (\hat{\boldsymbol{w}} - \boldsymbol{w}) + \frac{\alpha}{2}\|\hat{\boldsymbol{w}} - \boldsymbol{w}\|^2 \tag{D.100}$$

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

$$\begin{aligned}\forall \hat{\boldsymbol{w}} \in \mathbb{R}^p,\ \ell_z(\hat{\boldsymbol{w}}) &\geq \ell_z(\boldsymbol{w}) + \nabla \ell_z(\boldsymbol{w})^\top (\hat{\boldsymbol{w}} - \boldsymbol{w}) + \frac{\alpha}{2}\|\hat{\boldsymbol{w}} - \boldsymbol{w}\|^2 \\ &\geq \ell_z(\boldsymbol{w}) - \frac{1}{2\alpha}\|\nabla \ell_z(\boldsymbol{w})\|^2\end{aligned} \tag{D.101}$$

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

$$\ell_z(\boldsymbol{w}) - \ell_z(\underline{\mathbf{w}}) \leq \frac{1}{2\alpha}\|\nabla \ell_z(\boldsymbol{w})\|^2 \tag{D.102}$$

Therefore we can write:

$$\frac{\ell_z(\boldsymbol{w}) - \underline{\mathrm{B}}}{\|\nabla \ell_z(\boldsymbol{w})\|^2 + \delta} \leq \frac{\ell_z(\boldsymbol{w}) - \ell_z(\underline{\mathbf{w}}) + \varepsilon}{\|\nabla \ell_z(\boldsymbol{w})\|^2 + \delta} \leq \frac{\frac{1}{2\alpha}\|\nabla \ell_z(\boldsymbol{w})\|^2 + \varepsilon}{\|\nabla \ell_z(\boldsymbol{w})\|^2 + \delta}. \tag{D.103}$$

We introduce the function $\psi : x \in \mathbb{R}^+ \mapsto \frac{\frac{1}{2\alpha}x + \varepsilon}{x + \delta}$, and we compute its derivative:

$$\begin{aligned} \psi'(x) &= \frac{\frac{1}{2\alpha}(x+\delta) - \frac{1}{2\alpha}x - \varepsilon}{(x+\delta)^2}, \\ &= \frac{\frac{\delta}{2\alpha} - \varepsilon}{(x+\delta)^2} \geq 0. \quad (\delta \geq 2\alpha\varepsilon) \end{aligned} \tag{D.104}$$

Therefore ψ is monotonically increasing. As a result, we have:

$$\forall\ x \in \mathbb{R}^+,\ \psi(x) \leq \lim_{x\to\infty} \psi(x) = \frac{1}{2\alpha}. \tag{D.105}$$

Therefore we have that:

$$\frac{\frac{1}{2\alpha}\|\nabla \ell_z(\boldsymbol{w})\|^2 + \varepsilon}{\|\nabla \ell_z(\boldsymbol{w})\|^2 + \delta} = \psi\left(\|\nabla \ell_z(\boldsymbol{w})\|^2\right) \leq \frac{1}{2\alpha}, \tag{D.106}$$

which concludes the proof.

□

Theorem 9. *[Strongly Convex and Smooth] We assume that $\mathcal{X}$ is a convex set, and that for every $z \in \mathcal{Z}$, ℓ_z is α-strongly convex and β-smooth. Let $\underline{\mathrm{B}}$ be a uniform lower bound on $(\mathcal{P})$ and $\mathbf{w}_\star$ be a solution of $(\mathcal{P})$. Further suppose that $\mathbf{w}_\star$ is an ε-interpolation for $((\mathcal{P}), \underline{\mathrm{B}})$, and that $\delta > 2\beta\varepsilon$. Then ALI-G$^\infty$ applied to f satisfies:*

$$f(\boldsymbol{w}_{T+1}) - f_\star \leq \beta \exp\left(-\frac{\alpha T}{8\beta}\right)\|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 + \frac{2\delta}{\alpha} + \left(10\frac{\beta}{\alpha} + 4\frac{\beta^2}{\alpha^2}\right)\varepsilon. \tag{D.11}$$

In other words, f approximately converges to $f_\star$ at a rate of $\mathcal{O}(\exp(-\alpha T/8\beta))$.

Proof.

We condition the update on z_t drawn at random. The beginning of the proof is identical to that of Theorem 6 (and in particular requires $\delta > 2\beta\varepsilon$). In addition, we remark that $\delta > 2\beta\varepsilon \geq 2\alpha\varepsilon$, because it always holds true that $\beta \geq \alpha$. Combining inequalities (D.69) and (D.73), we obtain:

$$\begin{aligned} &\|\boldsymbol{w}_{t+1} - \mathbf{w}_\star\|^2 \\ &\quad \leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \frac{1}{2\beta}(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \frac{\delta}{4\beta^2} + \gamma_t(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}}), \end{aligned}$$

$$\begin{aligned}
&\leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \frac{1}{2\beta}(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \frac{\delta}{4\beta^2} + \gamma_t\varepsilon, \quad \text{(definition of } \varepsilon) \\
&\leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \frac{1}{2\beta}(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \frac{\delta}{4\beta^2} + \frac{\varepsilon}{2\alpha}. \quad \text{(Lemma 13)} \quad \text{(D.107)}
\end{aligned}$$

Let $\underline{\mathbf{w}}^{(z_t)}$ be the minimizer of ℓ_{z_t} on its unconstrained domain $\mathbb{R}^p$ (its existence is guaranteed by the strong convexity property). Then we exploit strong convexity to lower bound the progress made:

$$\begin{aligned}
&kern - 1em\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) \\
&= \ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\underline{\mathbf{w}}^{(z_t)}) + \ell_{z_t}(\mathbf{w}_\star) - \ell_{z_t}(\underline{\mathbf{w}}^{(z_t)}) - 2(\ell_{z_t}(\mathbf{w}_\star) - \ell_{z_t}(\underline{\mathbf{w}}^{(z_t)})) \\
&\geq \ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\underline{\mathbf{w}}^{(z_t)}) + \ell_{z_t}(\mathbf{w}_\star) - \ell_{z_t}(\underline{\mathbf{w}}^{(z_t)}) - 2(\ell_{z_t}(\mathbf{w}_\star) - \underline{\mathrm{B}}) \\
&\quad \text{(because } \ell_{z_t}(\underline{\mathbf{w}}^{(z_t)}) \geq \underline{\mathrm{B}}) \\
&\geq \ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\underline{\mathbf{w}}^{(z_t)}) + \ell_{z_t}(\mathbf{w}_\star) - \ell_{z_t}(\underline{\mathbf{w}}^{(z_t)}) - 2\varepsilon \quad \text{(definition of } \varepsilon) \\
&\geq \frac{\alpha}{2}\|\boldsymbol{w}_t - \underline{\mathbf{w}}^{(z_t)}\|^2 + \frac{\alpha}{2}\|\mathbf{w}_\star - \underline{\mathbf{w}}^{(z_t)}\|^2 - 2\varepsilon \quad (\alpha\text{-strong convexity}) \\
&\geq \frac{\alpha}{4}\|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - 2\varepsilon \quad \text{(Lemma 12)} \qquad \text{(D.108)}
\end{aligned}$$

We now combine inequalities (D.107) and (D.108):

$$\|\boldsymbol{w}_{t+1} - \mathbf{w}_\star\|^2 \leq \left(1 - \frac{\alpha}{8\beta}\right)\|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 + \frac{\varepsilon}{\beta} + \frac{\delta}{4\beta^2} + \frac{\varepsilon}{2\alpha} \qquad \text{(D.109)}$$

We use a trivial induction over t and write:

$$\begin{aligned}
&\|\boldsymbol{w}_{t+1} - \mathbf{w}_\star\|^2 \\
&\quad \leq \left(1 - \frac{\alpha}{8\beta}\right)\|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 + \frac{\varepsilon}{\beta} + \frac{\delta}{4\beta^2} + \frac{\varepsilon}{2\alpha}, \\
&\quad \leq \left(1 - \frac{\alpha}{8\beta}\right)^t \|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 + \sum_{k=0}^{t}\left(1 - \frac{\alpha}{8\beta}\right)^{t-k}\left(\frac{\varepsilon}{\beta} + \frac{\delta}{4\beta^2} + \frac{\varepsilon}{2\alpha}\right), \\
&\quad \leq \left(1 - \frac{\alpha}{8\beta}\right)^t \|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 + \sum_{k=0}^{\infty}\left(1 - \frac{\alpha}{8\beta}\right)^{k}\left(\frac{\varepsilon}{\beta} + \frac{\delta}{4\beta^2} + \frac{\varepsilon}{2\alpha}\right), \\
&\quad = \left(1 - \frac{\alpha}{8\beta}\right)^t \|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 + \frac{1}{\frac{\alpha}{8\beta}}\left(\frac{\varepsilon}{\beta} + \frac{\delta}{4\beta^2} + \frac{\varepsilon}{2\alpha}\right), \\
&\quad = \left(1 - \frac{\alpha}{8\beta}\right)^t \|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 + \frac{8\beta}{\alpha}\left(\frac{\varepsilon}{\beta} + \frac{\delta}{4\beta^2} + \frac{\varepsilon}{2\alpha}\right). \qquad \text{(D.110)}
\end{aligned}$$

In particular, we remark that the right hand-side of the equation is independent on z_t.

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

Since each ℓ_z is α-strongly convex and β-smooth, so is $f = \mathbb{E}_z[\ell_z]$. We introduce $\underline{\mathbf{w}}$ the minimizer of f on its unconstrained domain $\mathbb{R}^p$. Then we can write that for any $\boldsymbol{w} \in \mathbb{R}^p$:

$$
\begin{aligned}
&f(\boldsymbol{w}) - f(\mathbf{w}_\star) \\
&\quad \leq f(\boldsymbol{w}) - f(\underline{\mathbf{w}}), \quad (f(\underline{\mathbf{w}}) \leq f(\mathbf{w}_\star)) \\
&\quad \leq \nabla f(\underline{\mathbf{w}})^\top (\boldsymbol{w} - \underline{\mathbf{w}}) + \frac{\beta}{2}\|\boldsymbol{w} - \underline{\mathbf{w}}\|^2, \quad (f \text{ is } \beta\text{-smooth}) \\
&\quad = \frac{\beta}{2}\|\boldsymbol{w} - \underline{\mathbf{w}}\|^2, \quad (\nabla f(\underline{\mathbf{w}}) = \mathbf{0}) \\
&\quad \leq \beta(\|\boldsymbol{w} - \mathbf{w}_\star\|^2 + \|\mathbf{w}_\star - \underline{\mathbf{w}}\|^2), \quad \text{(Lemma 12)} \\
&\quad \leq \beta\|\boldsymbol{w} - \mathbf{w}_\star\|^2 + \frac{2\beta}{\alpha}\left(f(\mathbf{w}_\star) - f(\underline{\mathbf{w}})\right), \quad (f \text{ is } \alpha\text{-strongly convex}) \\
&\quad \leq \beta\|\boldsymbol{w} - \mathbf{w}_\star\|^2 + \frac{2\beta}{\alpha}\left(f(\mathbf{w}_\star) - \underline{\mathrm{B}}\right), \quad (\underline{\mathrm{B}} \leq f(\underline{\mathbf{w}}) \text{ by definition}) \\
&\quad \leq \beta\|\boldsymbol{w} - \mathbf{w}_\star\|^2 + 2\frac{\beta\varepsilon}{\alpha}, \quad \text{(definition of } \varepsilon) \qquad \text{(D.111)}
\end{aligned}
$$

We combine the results to obtain the final result:

$$
\begin{aligned}
&f(\boldsymbol{w}_{t+1}) - f(\mathbf{w}_\star) \\
&\quad \leq \beta\|\boldsymbol{w}_{t+1} - \mathbf{w}_\star\|^2 + 2\frac{\beta\varepsilon}{\alpha}, \\
&\quad \leq \beta\left(\left(1 - \frac{\alpha}{8\beta}\right)^t \|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 + \frac{8\beta}{\alpha}\left(\frac{\varepsilon}{\beta} + \frac{\delta}{4\beta^2} + \frac{\varepsilon}{2\alpha}\right)\right) + 2\frac{\beta\varepsilon}{\alpha}, \\
&\quad = \beta\left(1 - \frac{\alpha}{8\beta}\right)^t \|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 + \frac{8\beta}{\alpha}\left(\varepsilon + \frac{\delta}{4\beta} + \frac{\varepsilon\beta}{2\alpha}\right) + 2\frac{\beta\varepsilon}{\alpha}, \\
&\quad = \beta\left(1 - \frac{\alpha}{8\beta}\right)^t \|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 + \frac{2\delta}{\alpha} + \left(10\frac{\beta}{\alpha} + 4\frac{\beta^2}{\alpha^2}\right)\varepsilon, \\
&\quad \leq \beta\exp\left(-\frac{\alpha t}{8\beta}\right)\|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 + \frac{2\delta}{\alpha} + \left(10\frac{\beta}{\alpha} + 4\frac{\beta^2}{\alpha^2}\right)\varepsilon. \qquad \text{(D.112)}
\end{aligned}
$$

□

D.4.10 Theorem 10

Theorem 10. *We assume that $\mathcal{X}$ is a convex set, and that for every $z \in \mathcal{Z}$, ℓ_z is α-strongly convex and β-smooth. Let $\mathbf{w}_\star$ be an ε-interpolation for ($(\mathcal{P})$, $\underline{B}$), and suppose that $\delta > 2\beta\varepsilon$. Further assume that $\eta \geq \frac{1}{2\beta}$. Then if we apply ALI-G with a maximal learning-rate of η to f, we have:*

$$f(\boldsymbol{w}_{T+1}) - f_\star \leq \beta \exp\left(-\frac{\alpha T}{8\beta}\right) \|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 + \frac{2\delta}{\alpha} + \left(10\frac{\beta}{\alpha} + 4\frac{\beta^2}{\alpha^2}\right)\varepsilon. \tag{D.12}$$

Proof. Re-using inequalities (D.79) and (D.84) from the proof of Theorem 7, we obtain:

$$\|\boldsymbol{w}_{t+1} - \mathbf{w}_\star\|^2 \leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \frac{1}{2\beta}(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \frac{\delta}{4\beta^2} + \gamma_t(\ell_{z_t}(\mathbf{w}_\star) - \underline{B}). \tag{D.113}$$

This is exactly the same result as the first line of the inequality (D.107) in the proof of Theorem 9. Then the rest of the proof is identical to the one of Theorem 9. □

D.4.11 Theorem 11

Theorem 11. *We assume that $\mathcal{X}$ is a convex set, and that for every $z \in \mathcal{Z}$, ℓ_z is α-strongly convex and β-smooth. Let $\mathbf{w}_\star$ be an ε-interpolation for ($(\mathcal{P})$, $\underline{B}$), and suppose that $\delta > 2\beta\varepsilon$. Further assume that $\eta \leq \frac{1}{2\beta}$. Then if we apply ALI-G with a maximal learning-rate of η to f, we have:*

$$f(\boldsymbol{w}_{T+1}) - f_\star \leq \beta \exp\left(-\frac{\alpha\eta T}{4}\right) \|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 + \frac{2\delta}{\alpha} + \frac{14\varepsilon\beta}{\alpha}. \tag{D.13}$$

Proof. Re-using inequalities (D.87) and (D.93) from the proof of Theorem 8, we can write:

$$\begin{aligned}
&\|\boldsymbol{w}_{t+1} - \mathbf{w}_\star\|^2 \\
&\qquad \leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \eta(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \frac{\eta\delta}{2\beta} + \gamma_t(\ell_{z_t}(\mathbf{w}_\star) - \underline{B}), \\
&\qquad \leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \eta(\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star)) + \frac{\eta\delta}{2\beta} + \eta\varepsilon \\
&\qquad\quad \text{(using } \gamma_t \leq \eta,\ 0 \leq \ell_{z_t}(\mathbf{w}_\star) - \underline{B} \leq \varepsilon).
\end{aligned} \tag{D.114}$$

Furthermore, the inequality (D.108) gives:

$$\ell_{z_t}(\boldsymbol{w}_t) - \ell_{z_t}(\mathbf{w}_\star) \geq \frac{\alpha}{4}\|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - 2\varepsilon \tag{D.115}$$

Therefore, we can write:

$$\begin{aligned}\|\boldsymbol{w}_{t+1} - \mathbf{w}_\star\|^2 &\leq \|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 - \frac{\alpha\eta}{4}\|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 + \frac{\eta\delta}{2\beta} + 3\eta\varepsilon, \\ &= \left(1 - \frac{\alpha\eta}{4}\right)\|\boldsymbol{w}_t - \mathbf{w}_\star\|^2 + \frac{\eta\delta}{2\beta} + 3\eta\varepsilon.\end{aligned} \tag{D.116}$$

Then a trivial induction gives that:

$$\begin{aligned}\|\boldsymbol{w}_{T+1} - \mathbf{w}_\star\|^2 &\leq \left(1 - \frac{\alpha\eta}{4}\right)^T \|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 + \left(\frac{\eta\delta}{2\beta} + 3\eta\varepsilon\right)\sum_{t=0}^{T}\left(1 - \frac{\alpha\eta}{4}\right)^t, \\ &\leq \left(1 - \frac{\alpha\eta}{4}\right)^T \|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 + \left(\frac{\eta\delta}{2\beta} + 3\eta\varepsilon\right)\sum_{t=0}^{\infty}\left(1 - \frac{\alpha\eta}{4}\right)^t, \\ &= \left(1 - \frac{\alpha\eta}{4}\right)^T \|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 + \left(\frac{\eta\delta}{2\beta} + 3\eta\varepsilon\right)\frac{1}{1 - \left(1 - \frac{\alpha\eta}{4}\right)}, \\ &= \left(1 - \frac{\alpha\eta}{4}\right)^T \|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 + \frac{2\delta}{\alpha\beta} + \frac{12\varepsilon}{\alpha}.\end{aligned} \tag{D.117}$$

We now re-use the inequality (D.111) to write:

$$\begin{aligned}f(\boldsymbol{w}_{T+1}) - f_\star &\leq \beta\|\boldsymbol{w}_{T+1} - \mathbf{w}_\star\|^2 + \frac{2\beta\varepsilon}{\alpha}, \\ &\leq \beta\left(1 - \frac{\alpha\eta}{4}\right)^T \|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 + \frac{2\delta}{\alpha} + \frac{14\varepsilon\beta}{\alpha}, \\ &\leq \beta\exp\left(\frac{-\alpha\eta T}{4}\right)\|\boldsymbol{w}_0 - \mathbf{w}_\star\|^2 + \frac{2\delta}{\alpha} + \frac{14\varepsilon\beta}{\alpha}.\end{aligned} \tag{D.118}$$

□

D.5 Additional Experimental Details

D.5.1 Standard Deviation of CIFAR Results

Task	Optimizer	Avg	Std
DN10	ADAMW	92.6	0.08
DN10	ALIG	95.0	0.16
DN10	AMSGRAD	91.7	0.25
DN10	DFW	94.6	0.22
DN10	L4ADAM	90.8	0.09
DN10	L4MOM	91.9	0.17
DN10	SGD	95.1	0.21
DN10	YOGI	92.1	0.38
DN100	ADAMW	69.5	0.54
DN100	ALIG	76.3	0.14
DN100	AMSGRAD	69.4	0.41
DN100	DFW	73.2	0.29
DN100	L4ADAM	60.5	0.64
DN100	L4MOM	62.6	1.98
DN100	SGD	76.3	0.22
DN100	YOGI	69.6	0.34
WRN10	ADAMW	92.1	0.34
WRN10	ALIG	95.2	0.09
WRN10	AMSGRAD	90.8	0.31
WRN10	DFW	94.2	0.19
WRN10	L4ADAM	90.5	0.09
WRN10	L4MOM	91.6	0.24
WRN10	SGD	95.3	0.31
WRN10	YOGI	91.2	0.27
WRN100	ADAMW	69.6	0.51
WRN100	ALIG	75.8	0.29
WRN100	AMSGRAD	68.7	0.70
WRN100	DFW	76.0	0.24
WRN100	L4ADAM	61.7	2.17
WRN100	L4MOM	61.4	0.86
WRN100	SGD	77.8	0.13
WRN100	YOGI	68.7	0.47

Table D.1: *Test Accuracy (%) on CIFAR including standard deviations. Each experiment was run three times.*

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Man, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Vi, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org.

Amos, B., Koltun, V., and Kolter, J. Z. (2019). The limited multi-label projection layer. *arXiv preprint*.

Amos, B., Xu, L., and Kolter, J. Z. (2017). Input convex neural networks. *International Conference on Machine Learning*.

Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., and De Freitas, N. (2016). Learning to learn by gradient descent by gradient descent. *Neural Information Processing Systems*.

Arpit, D., Jastrzębski, S., Ballas, N., Krueger, D., Bengio, E., Kanwal, M. S., Maharaj, T., Fischer, A., Courville, A., Bengio, Y., and Lacoste-Julien, S. (2017). A closer look at memorization in deep networks. *International Conference on Machine Learning*.

Asi, H. and Duchi, J. C. (2019). Stochastic (approximate) proximal point methods: Convergence, optimality, and adaptivity. *SIAM Journal on Optimization*.

Ba, J., Grosse, R., and Martens, J. (2017). Distributed second-order optimization using kronecker-factored approximations. *International Conference on Learning Representations*.

Bach, F. (2015). Duality between subgradient and conditional gradient methods. *SIAM Journal on Optimization*.

Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I. J., Bergeron, A., Bouchard, N., and Bengio, Y. (2012). *Theano: new features and speed improvements*.

Baydin, A. G., Cornish, R., Rubio, D. M., Schmidt, M., and Wood, F. (2018). Online learning rate adaptation with hypergradient descent. *International Conference on Learning Representations*.

Beck, A. and Teboulle, M. (2012). Smoothing and first order methods: A unified framework. *SIAM Journal on Optimization*.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. *Neural Information Processing Systems*.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: a CPU and GPU math expression compiler. *Python for Scientific Computing Conference (SciPy)*.

Bernstein, J., Wang, Y.-X., Azizzadenesheli, K., and Anandkumar, A. (2018). signsgd: Compressed optimisation for non-convex problems. *International Conference on Machine Learning*.

Berrada, L., Zisserman, A., and Kumar, M. P. (2017). Trusting SVM for piecewise linear CNNs. *International Conference on Learning Representations*.

Berrada, L., Zisserman, A., and Kumar, M. P. (2018). Smooth loss functions for deep top-k classification. *International Conference on Learning Representations*.

Berrada, L., Zisserman, A., and Kumar, M. P. (2019a). Deep Frank-Wolfe for neural network optimization. *International Conference on Learning Representations*.

Berrada, L., Zisserman, A., and Kumar, M. P. (2019b). Training neural networks for and by interpolation. *arXiv preprint*.

Bertsekas, D. P. (2003). *Convex Analysis and Optimization*. Athena Scientific.

Bietti, A., Mialon, G., Chen, D., and Mairal, J. (2019). A kernel perspective for regularizing deep neural networks. *International Conference on Machine Learning*.

Binder, A., Müller, K.-R., and Kawanabe, M. (2012). On taxonomies for multi-class image categorization. *International Journal of Computer Vision*.

Blaschko, M. B. and Lampert, C. H. (2008). Learning to localize objects with structured output regression. *European Conference on Computer Vision*.

Botev, A., Ritter, H., and Barber, D. (2017). Practical gauss-newton optimisation for deep learning. *International Conference on Machine Learning*.

Bowman, S. R., Angeli, G., Potts, C., and Manning, C. D. (2015). A large annotated corpus for learning natural language inference. *Conference on Empirical Methods in Natural Language Processing*.

Brännlund, U. (1995). A generalized subgradient method with relaxation step. *Mathematical Programming*.

Bubeck, S. (2015). Convex optimization: Algorithms and complexity. *Foundations and Trends in Machine Learning*.

Cai, L. and Hofmann, T. (2004). Hierarchical document categorization with support vector machines. *Conference on Information and Knowledge Management*.

Chang, X., Yu, Y.-L., and Yang, Y. (2017). Robust top-k multiclass SVM for visual category recognition. *International Conference on Knowledge Discovery and Data Mining*.

Chaudhari, P., Choromanska, A., Soatto, S., and LeCun, Y. (2017). Entropy-SGD: Biasing gradient descent into wide valleys. *International Conference on Learning Representations*.

Chaudhari, P. and Soatto, S. (2018). Stochastic gradient descent performs variational inference, converges to limit cycles for deep networks. *International Conference on Learning Representations*.

Chen, J. and Gu, Q. (2018). Padam: Closing the generalization gap of adaptive gradient methods in training deep neural networks. *arXiv preprint*.

Chen, X., Liu, S., Sun, R., and Hong, M. (2019). On the convergence of a class of adam-type algorithms for non-convex optimization. *International Conference on Learning Representations*.

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2016). Fast and accurate deep network learning by exponential linear units (ELUs). *International Conference on Learning Representations*.

Conneau, A., Kiela, D., Schwenk, H., Barrault, L., and Bordes, A. (2017). Supervised learning of universal sentence representations from natural language inference data. *Conference on Empirical Methods in Natural Language Processing*.

Crammer, K. and Singer, Y. (2001). On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*.

Défossez, A. and Bach, F. (2017). Adabatch: Efficient gradient aggregation rules for sequential and parallel stochastic gradient methods. *arXiv preprint*.

Desjardins, G., Simonyan, K., Pascanu, R., , and Kavukcuoglu, K. (2015). Natural neural networks. *Neural Information Processing Systems*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. *Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2010). The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*.

Fan, Y., Lyu, S., Ying, Y., and Hu, B.-G. (2017). Learning with average top-k loss. *Neural Information Processing Systems*.

Frank, M. and Wolfe, P. (1956). An algorithm for quadratic programming. *Naval Research Logistics Quarterly*.

Frerix, T., Möllenhoff, T., Moeller, M., and Cremers, D. (2018). Proximal backpropagation. *International Conference on Learning Representations*.

Friedman, J., Hastie, T., and Tibshirani, R. (2001). *The elements of statistical learning*. Springer series in statistics New York.

Goel, S., Kanade, V., Klivans, A., and Thaler, J. (2017). Reliably learning the ReLU in polynomial time. *Conference on Learning Theory*.

Goodfellow, I. J., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S. G., Grefenstette, E., Ramalho, T., Agapiou, J., Badia, A. P., Hermann, K. M., Zwols, Y., Ostrovski, G., Cain, A., King, H., Summerfield, C., Blunsom, P., Kavukcuoglu, K., and Hassabis, D. (2016). Hybrid computing using a neural network with dynamic external memory. *Nature*.

Grosse, R. and Martens, J. (2016). A kronecker-factored approximate fisher matrix for convolution layers. *International Conference on Machine Learning*.

Gulcehre, C., Moczulski, M., Visin, F., and Bengio, Y. (2017). Mollifying networks. *International Conference on Learning Representations*.

Hardt, M., Recht, B., and Singer, Y. (2016). Train faster, generalize better: Stability of stochastic gradient descent. *International Conference on Machine Learning*.

Hazan, T., Keshet, J., and McAllester, D. A. (2010). Direct loss minimization for structured prediction. *Neural Information Processing Systems*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. *Conference on Computer Vision and Pattern Recognition*.

Heinemann, U., Livni, R., Eban, E., Elidan, G., and Globerson, A. (2016). Improper deep kernels. *International Conference on Artificial Intelligence and Statistics*.

Henriques, J. F., Ehrhardt, S., Albanie, S., and Vedaldi, A. (2019). Small steps and giant leaps: Minimal newton solvers for deep learning. *International Conference on Computer Vision*.

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*.

Hochreiter, S. and Obermayer, K. (2005). Optimal gradient-based learning using importance weights. *International Joint Conference on Neural Networks*.

Hoffer, E., Hubara, I., and Soudry, D. (2017). Train longer, generalize better: closing the generalization gap in large batch training of neural networks. *Neural Information Processing Systems*.

Horst, R. and Thoai, N. V. (1999). DC programming: overview. *Journal of Optimization Theory and Applications*.

Huang, G., Liu, Z., Weinberger, K. Q., and van der Maaten, L. (2017). Densely connected convolutional networks. *Conference on Computer Vision and Pattern Recognition*.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International Conference on Machine Learning*.

Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., Fernando, C., and Kavukcuoglu, K. (2017). Population based training of neural networks. *arXiv preprint*.

Jiang, H., Graillat, S., Barrio, R., and Yang, C. (2016). Accurate, validated and fast evaluation of elementary symmetric functions and its application. *Applied Mathematics and Computation*.

Joachims, T., Finley, T., and Yu, C.-N. J. (2009). Cutting-plane training of structural SVMs. *Journal of Machine Learning Research*.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. *Technical Report*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*.

Lacoste-Julien, S. and Jaggi, M. (2013). Block-coordinate Frank-Wolfe optimization for structural SVMs. *International Conference on Machine Learning*.

Lapin, M., Hein, M., and Schiele, B. (2015). Top-k multiclass SVM. *Neural Information Processing Systems*.

Lapin, M., Hein, M., and Schiele, B. (2016). Loss functions for top-k error: Analysis and insights. *Conference on Computer Vision and Pattern Recognition*.

Lapin, M., Hein, M., and Schiele, B. (2017). Analysis and optimization of loss functions for multiclass, top-k, and multilabel classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*.

Lee, D.-H., Zhang, S., Fischer, A., and Bengio, Y. (2015). Difference target propagation. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*.

Lee, Y.-J. and Mangasarian, O. L. (2001). SSVM: A smooth support vector machine for classification. *Computational optimization and Applications*.

Levy, K. (2017). Online to offline conversions, universality and adaptive minibatch sizes. *Neural Information Processing Systems*.

Lewis, A. S. and Wright, S. J. (2016). A proximal method for composite minimization. *Mathematical Programming*.

Li, K. and Malik, J. (2017). Learning to optimize. *International Conference on Learning Representations*.

Li, X. and Orabona, F. (2019). On the convergence of stochastic gradient descent with adaptive stepsizes. *International Conference on Artificial Intelligence and Statistics*.

Li, Y., Song, Y., and Luo, J. (2017). Improving pairwise ranking for multi-label image classification. *Conference on Computer Vision and Pattern Recognition*.

Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J., and Han, J. (2019). On the variance of the adaptive learning rate and beyond. *arXiv preprint*.

Locatello, F., Khanna, R., Tschannen, M., and Jaggi, M. (2017). A unified optimization view on generalized matching pursuit and frank-wolfe. *International Conference on Artificial Intelligence and Statistics*.

Loshchilov, I. and Hutter, F. (2017). SGDR: Stochastic gradient descent with warm restarts. *International Conference on Learning Representations*.

Loshchilov, I. and Hutter, F. (2019). Fixing weight decay regularization in adam. *International Conference on Learning Representations*.

Luo, L., Xiong, Y., Liu, Y., and Sun, X. (2019). Adaptive gradient methods with dynamic bound of learning rate. *International Conference on Learning Representations*.

Ma, S., Bassily, R., and Belkin, M. (2018). The power of interpolation: Understanding the effectiveness of sgd in modern over-parametrized learning. *International Conference on Machine Learning*.

Mahajan, D., Girshick, R., Ramanathan, V., He, K., Paluri, M., Li, Y., Bharambe, A., and van der Maaten, L. (2018). Exploring the limits of weakly supervised pretraining. *European Conference on Computer Vision*.

Martens, J., Ba, J., and Johnson, M. (2018). Kronecker-factored curvature approximations for recurrent neural networks. *International Conference on Learning Representations*.

Martens, J. and Grosse, R. (2015). Optimizing neural networks with Kronecker-factored approximate curvature. *International Conference on Machine Learning*.

Martens, J. and Sutskever, I. (2012). Training deep and recurrent networks with Hessian-free optimization. *Neural Networks: Tricks of the Trade*.

Melzer, D. (1986). On the expressibility of piecewise-linear continuous functions as the difference of two piecewise-linear convex functions. In *Quasidifferential Calculus*. Springer.

Mohapatra, P., Dokania, P., Jawahar, C. V., and Kumar, M. P. (2016). Partial linearization based optimization for multi-class SVM. *European Conference on Computer Vision*.

Mukkamala, M. C. and Hein, M. (2017). Variants of rmsprop and adagrad with logarithmic regret bounds. *International Conference on Machine Learning*.

Nedić, A. and Bertsekas, D. (2001a). Convergence rate of incremental subgradient algorithms. *Stochastic optimization: algorithms and applications*.

Nedić, A. and Bertsekas, D. (2001b). Incremental subgradient methods for nondifferentiable optimization. *SIAM Journal on Optimization*.

Nesterov, Y. (1983). A method of solving a convex programming problem with convergence rate $\mathcal{O}(1/k^2)$. *Soviet Mathematics Doklady*.

Neyshabur, B., Bhojanapalli, S., McAllester, D., and Srebro, N. (2017). Exploring generalization in deep learning. *Neural Information Processing Systems*.

Neyshabur, B., Salakhutdinov, R. R., and Srebro, N. (2015). Path-sgd: Path-normalized optimization in deep neural networks. *Neural Information Processing Systems*.

Neyshabur, B., Wu, Y., Salakhutdinov, R. R., and Srebro, N. (2016). Path-normalized optimization of recurrent neural networks with relu activations. *Neural Information Processing Systems*.

Nocedal, J. and Wright, S. (2006). *Numerical optimization*. Springer Science & Business Media.

Norouzzadeh, M. S., Nguyen, A., Kosmala, M., Swanson, A., Palmer, M. S., Packer, C., and Clune, J. (2018). Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning. *Proceedings of the National Academy of Sciences*.

Oberman, A. M. and Prazeres, M. (2019). Stochastic gradient descent with polyak's learning rate. *arXiv preprint*.

Ollivier, Y. (2013). Riemannian metrics for neural networks. *Information and Inference: a Journal of the IMA*.

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

Orabona, F. and Pál, D. (2015). Scale-free algorithms for online linear optimization. *International Conference on Algorithmic Learning Theory*.

Osokin, A., Alayrac, J.-B., Lukasewitz, I., Dokania, P., and Lacoste-Julien, S. (2016). Minding the gaps for block Frank-Wolfe optimization of structured SVMs. *International Conference on Machine Learning*.

Parikh, N. and Boyd, S. (2014). Proximal algorithms. *Foundations and Trends in Optimization*.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch. *NIPS Autodiff Workshop*.

Polyak, B. T. (1969). Minimization of unsmooth functionals. *USSR Computational Mathematics and Mathematical Physics*.

Ravi, S. and Larochelle, H. (2017). Optimization as a model for few-shot learning. *International Conference on Learning Representations*.

Reddi, S. J., Kale, S., and Kumar, S. (2018). On the convergence of adam and beyond. *International Conference on Learning Representations*.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*.

Rolinek, M. and Martius, G. (2018). L4: Practical loss-based stepsize adaptation for deep learning. *Neural Information Processing Systems*.

Roux, N. L., Manzagol, P.-A., and Bengio, Y. (2008). Topmoumoute online natural gradient algorithm. *Neural Information Processing Systems*.

Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning representations by back-propagating errors. *Nature*.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*.

Schaul, T., Zhang, S., and LeCun, Y. (2013). No more pesky learning rates. *International Conference on Machine Learning*.

Schneider, F., Balles, L., and Hennig, P. (2019). DeepOBS: A deep learning optimizer benchmark suite. *International Conference on Learning Representations*.

Schwing, A. G., Hazan, T., Pollefeys, M., and Urtasun, R. (2012). Efficient structured prediction with latent variables for general graphical models. *International Conference on Machine Learning*.

Shah, N., Kolmogorov, V., and Lampert, C. H. (2015). A multi-plane block-coordinate Frank-Wolfe algorithm for training structural SVMs with a costly max-oracle. *Conference on Computer Vision and Pattern Recognition*.

Shalev-Shwartz, S., Singer, Y., and Srebro, N. (2009). Pegasos: Primal estimated sub-gradient solver for SVM. *International Conference on Machine Learning*.

Shalev-Shwartz, S. and Zhang, T. (2016). Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization. *Mathematical Programming*.

Shazeer, N. and Stern, M. (2018). Adafactor: Adaptive learning rates with sublinear memory cost. *International Conference on Machine Learning*.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. (2019). Megatron-lm: Training multi-billion parameter language models using gpu model parallelism. *arXiv preprint*.

Shor, N. Z. (1985). *Minimization methods for non-differentiable functions*. Springer Series in Computational Mathematics.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations*.

Singh, G. and Shawe-Taylor, J. (2018). Faster convergence & generalization in DNNs. *arXiv preprint*.

Song, Y., Schwing, A. G., Zemel, R., and Urtasun, R. (2016). Training deep neural networks via direct loss minimization. *International Conference on Machine Learning*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. *Conference on Computer Vision and Pattern Recognition*.

Tan, C., Ma, S., Dai, Y.-H., and Qian, Y. (2016). Barzilai-borwein step size for stochastic gradient descent. *Neural Information Processing Systems*.

Taskar, B., Guestrin, C., and Koller, D. (2003). Max-margin Markov networks. *Neural Information Processing Systems*.

Taylor, G., Burmeister, R., Xu, Z., Singh, B., Patel, A., and Goldstein, T. (2016). Training neural networks without gradients: A scalable ADMM approach. *International Conference on Machine Learning*.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*.

Tomavsev, N., Glorot, X., Rae, J. W., Zielinski, M., Askham, H., Saraiva, A., Mottram, A., Meyer, C., Ravuri, S., Protsyuk, I., Connell, A., Hughes, C. O., Karthikesalingam, A., Cornebise, J., Montgomery, H., Rees, G., Laing, C., Baker, C. R., Peterson, K., Reeves, R., Hassabis, D., King, D., Suleyman, M., Back, T., Nielson, C., Ledsam, J. R., and Mohamed, S. (2019). A clinically applicable approach to continuous prediction of future acute kidney injury. *Nature*.

Townsend, J. (2017). A new trick for calculating jacobian vector products. `https://j-towns.github.io/2017/06/12/A-new-trick.html`. Accessed: 2019-11-27.

Tsochantaridis, I., Hofmann, T., Joachims, T., and Altun, Y. (2004). Support vector machine learning for interdependent and structured output spaces. *International Conference on Machine Learning*.

Vaswani, S., Bach, F., and Schmidt, M. (2019a). Fast and faster convergence of sgd for over-parameterized models and an accelerated perceptron. *International Conference on Artificial Intelligence and Statistics*.

Vaswani, S., Mishkin, A., Laradji, I., Schmidt, M., Gidel, G., and Lacoste-Julien, S. (2019b). Painless stochastic gradient: Interpolation, line-search, and convergence rates. *arXiv preprint*.

Wichrowska, O., Maheswaranathan, N., Hoffman, M. W., Colmenarejo, S. G., Denil, M., de Freitas, N., and Sohl-Dickstein, J. (2017). Learned optimizers that scale and generalize. *International Conference on Machine Learning*.

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., and Recht, B. (2017). The marginal value of adaptive gradient methods in machine learning. *Neural Information Processing Systems*.

Wu, X., Ward, R., and Bottou, L. (2018). WNGrad: Learn the learning rate in gradient descent. *arXiv preprint*.

Yan, C., Luo, M., Liu, H., Li, Z., and Zheng, Q. (2017). Top-k multi-class svm using multiple features. *Information Sciences*.

Yu, C.-N. J. and Joachims, T. (2009). Learning structural SVMs with latent variables. *International Conference on Machine Learning*.

Yuille, A. L. and Rangarajan, A. (2002). The concave-convex procedure (CCCP). *Neural Information Processing Systems*.

Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. *British Machine Vision Conference*.

Zaheer, M., Reddi, S., Sachan, D., Kale, S., and Kumar, S. (2018). Adaptive methods for nonconvex optimization. *Neural Information Processing Systems*.

Zeiler, M. (2012). ADADELTA: an adaptive learning rate method. *arXiv preprint*.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2017a). Understanding deep learning requires rethinking generalization. *International Conference on Learning Representations*.

Zhang, G., Wang, C., Xu, B., and Grosse, R. (2018). Three mechanisms of weight decay regularization. *International Conference on Learning Representations*.

Zhang, Y., Liang, P., and Wainwright, M. J. (2017b). Convexified convolutional neural networks. *International Conference on Machine Learning*.

Zhang, Z., Wu, Y., and Wang, G. (2017c). Bpgrad: Towards global optimality in deep learning via branch and pruning. *Conference on Computer Vision and Pattern Recognition*.

Zheng, H., Yang, Z., Liu, W., Liang, J., and Li, Y. (2015). Improving deep neural networks using softplus units. *International Joint Conference on Neural Networks*.

Zheng, S. and Kwok, J. T. (2017). Follow the moving leader in deep learning. *International Conference on Machine Learning*.

Zhou, Y., Yang, J., Zhang, H., Liang, Y., and Tarokh, V. (2019). Sgd converges to global minimum in deep learning via star-convex path. *International Conference on Learning Representations*.