MASTER THESIS

Online Step Size Adaptation for Stochastic Optimization

A thesis submitted in partial fulfillment of the degree requirements

Master of Science

Graduate School of Neural Information Processing

Faculty of Science
Faculty of Medicine

Eberhard-Karls-Universität Tübingen

Presented by
Andrii Zadaianchuk
from Odessa, Ukraine

Tübingen
December 6, 2018

Declaration of Authorship

Thesis Advisor
Prof. Dr. Philipp Hennig
Department of
Computer Science

Second Reader
Prof. Dr. Martin A. Giese
Department of
Cognitive Neurology

I, Andrii Zadaianchuk, declare that this thesis titled, "Online Step Size Adaptation for Stochastic Optimization" and the work presented in it are my own. I confirm that:

- I have written the dissertation myself and have not used any sources and aids other than those indicated.
- I have not included data generated in one of my laboratory rotations and already presented in the respective laboratory report

Date: ____

Signature: ____

“*An approximate answer to the right problem is worth a good deal more than an exact answer to an approximate problem.*”

John Tukey

EBERHARD-KARLS-UNIVERSITÄT TÜBINGEN

Abstract

Faculty of Science
Faculty of Medicine
Graduate School of Neural Information Processing

Master of Science

Online Step Size Adaptation for Stochastic Optimization

by Andrii Zadaianchuk

An automatic finding of the optimal step sizes schedule for stochastic optimization algorithms is quite important for machine learning practitioners as it could save a lot of machine and human resources. Hypergradient descent (HD) is a recently rediscovered algorithm that adapts a step size using a gradient descent (GD) update of the loss function as a function of the step size. In this thesis, we provide an interpretation of HD as an iteration of the proximal point algorithm applied to a linear approximation of the loss function. However, it is possible to develop other adaptation rules using better approximations of the loss function. We develop proximal quadratic (PQ) adaptation that applies one iteration of a proximal point algorithm to the quadratic model of the loss function. The quadratic model is fitted by reusing the stochastic estimators of loss function values and its gradients from the previous iteration. The comparison between PQ and HD adaptations shows that the performance of the optimization algorithm with PQ adaptation is better than the performance of the algorithm with HD adaptation and the sensitivity of the step-size hyperparameters is smaller, so one can more easily tune step-size hyperparameters of the optimization algorithm with PQ adaptation.

Acknowledgements

I am deeply grateful to Prof. Dr. Philipp Henig, for giving me the opportunity to work in his group. I am thankful for the wisest guidance during all the stages of the research and writing of the thesis. I would like to extend my sincere thanks to Lukas Balles, with whom we actively collaborated during my master project and whose expertise and skills helped me to learn how to do proper high-quality research.

I am truly thankful to the Probabilistic Numerics group. Their feedback and discussion of the further steps helped me to see my project from different perspectives. In addition, they create the best environment for making first steps in research and motivate me to continue my road in academia.

I also sincerely thank Konstantin Mishenko, with whom we discussed the theoretical aspects of the project. His expertise in stochastic optimization and proximal algorithms helped me to find the right language to describe the process of step size adaptation.

I am thankful to all the people who supported me during these two years of intensive studies in Tübingen, including my groupmates, DAAD foundation, the GTC administration and the professors who helped me to develop my mathematical and coding skills as well as giving intuition about models that could describe the computations inside the brain.

My biggest thanks go to my family, who support me all this time. I'm so happy that I always have a place where I can recharge my energy to be able to do next steps. I want to thank my friends – Olga, Alex, Nastiia and many other who inspired me to do my best. With you, I see the power of friendship and it's huge effect on every aspect of life including studying and research. Thanks for the motivation to continue with an exciting and challenging path of scientific discovery. Thanks for spending hours of listening to my description of the project I'm working on and giving advices how to improve my thesis. Melissa and Sarah, thank you so much for careful proof-reading that helped to improve the quality of the thesis and my English skills as well. Finally, I'm so thankful to my partner, Kseniia, who supported me all this time and listened to my ideas helping me to find the right direction.

Contents

List of Figures

1 Introduction

Machine learning is the field that aims to transform data to intelligent algorithms which can predict characteristics of unseen data from the same process or make task-related decisions. There are many applications in industry where machine learning can be used to create new products and services or to optimize the workflow inside a company. Striking examples of new innovative products created with the use of machine learning algorithms are virtual personal assistants such as Siri from Apple that use speech recognition (Chiu et al., 2017) and natural language processing (Kumar et al., 2016) to simplify the control of the phone by voice. The positive impact of machine learning on organizational decisions has also been investigated by many researchers and companies. Delen et al., 2013 investigated the impact of machine learning algorithms on management, on organizational performance of small and medium-sized enterprises in the service industry. Following their investigation, they reached the conclusion that there is a strong and positive relationship between the usage of machine learning and increased organizational performance.

Besides direct application, machine learning makes an impact on many other fields. Scientists gain many valuable tools and ideas from machine learning. For theoreticians in several fields like computational neuroscience, machine learning models, such as probabilistic graphical models (Koller and Friedman, 2009) could be suitable for describing the processes of interest. It is also useful for data processing and data analysis. Machine learning has been used for approximation of molecules interaction (Brockherde et al., 2017), imaging techniques in neuroscience (Vogt, 2018), drug discovery (Lavecchia, 2015) and others. In addition to the use of machine learning as a data processing tool, it is possible to use it as a scientific method. For example, the ability to classify the feature of interest using different parts of the EEG time series can show when the information connected with the feature of interest was processed in the brain (King and Dehaene, 2014).

Depending on the nature of available data, there are several kinds of machine learning models. The central types are supervised, unsupervised and reinforcement learning. Reinforcement learning is used to actively learn how to behave in an unknown environment. In this setting, the data is the reward for taken actions. Unsupervised learning techniques uncover the structure of the dependencies between the different features in data. For instance, they could be used to learn a useful transformation of the data (e.g. a transformation that distinguishes between different interconnected parts of the data - clusters) that reveals this structure.

In supervised learning, one distinguishes a special group of features - the target features. In this framework, the conditional distribution over target futures given the input features is approximated by choosing from a family of distributions such that the available data is plausible under chosen distribution. Given target and input features, it is possible to calculate the loss function that measures how good the current performance of the algorithm is. The learning process is the minimization of the loss function. In our work, we focused on a supervised learning framework because results of optimization algorithms under this framework could then be compared to previously obtained results (Johnson and Zhang, 2013; Tan et al., 2016) that provide better optimization techniques using the structure of the supervised loss function.

In a supervised learning framework, there is a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ sampled from some unknown true distribution $P(x, y)$, where $x_i \in \mathbb{R}^n$ is the a feature vector and $y_i \in \mathbb{R}^m$ is an output vector. Our aim is to use this dataset to discover a function $h(x) : \mathbb{R}^n \to \mathbb{R}^m$ that predicts an output vector y_{new} given a feature vector x_{new} from $P(x)$.

1.1 Empirical Risk Minimization

The learning process in supervised learning is often formalized as an expected risk minimization over a set of functions $\mathcal{H}$. For large-scale machine learning the set $\mathcal{H}$ is usually represented as a family of parametrized functions $\mathcal{H} = \{h(x, \theta) : \theta \in \mathbb{R}^d\}$. In such a situation, a choice of the prediction function is fully determined by the choice of the vector of parameters θ. The expected risk is given by

$$R(\theta) = \int_{\mathbb{R}^n \times \mathbb{R}^m} l(h(x, \theta), y) dP(x, y). \tag{1.1}$$

where $P(x, y)$ is the true distribution over input-output pairs and $l(h(x, \theta), y)$ is a loss function that compares the predicted output $h(x, \theta)$ and the true output y. The choice of the loss function depends on the task of interest. For example, for the multi-class classification task, a cross-entropy loss is widely used. In classification, the predicted output $h(x, \theta)$ is often described as a concatenated vector of conditional distributions $h(x, \theta)_i = P(\mathcal{C}_i | x)$ over possible classes $\{\mathcal{C}_i\}_{i=1}^m$ given a feature vector x. If the information about the class is encoded in an one-hot vector $y \in \mathbb{R}^m$ then the cross-entropy loss function can be written as

$$l(h(x, \theta), y) = -\sum_{j=1}^{m} y_{(j)} \log h(x, \theta)_{(j)}. \tag{1.2}$$

Even if the loss function is defined in a manner that it represents our task adequately, one could not usually compute the expected risk, as the true distribution $P(x, y)$ is unknown. However, one can approximate it by the empirical distribution of our data and minimize an empirical risk instead of the expected risk. The empirical risk could be described as

$$R_{\text{emp}}(\theta) = \frac{1}{N}\sum_{i=1}^{N} l(h(x_i,\theta), y_i). \tag{1.3}$$

Substitution of the expected risk by its approximation could lead to overfitting of the model to the training data, thus, the loss function $l(h(x,\theta),y)$ is usually modified by the addition of the regularization term that helps to prevent overfitting. In this case, the learning process could be described as an unconstrained minimization problem

$$\min_{\theta} \mathcal{L}(\theta), \tag{1.4}$$

where the objective function $\mathcal{L}(\theta)$ is the sum of an empirical risk $R_{\text{emp}}(\theta)$ and the regularization term $\mathcal{L}_{\text{reg}}(\theta)$.

To verify that such a learning process minimizes the expected risk we could use a validation loss that is the empirical loss on data that was not used during training. For instance, cross-validation loss is used to tune the hyperparameters of the optimizer that trains a machine learning model (Snoek, Larochelle, and Adams, 2012).

1.2 Optimization Methods in Large-scale Machine Learning

Next, we shortly discuss commonly used optimization algorithms for the objective function $\mathcal{L}(\theta)$ given that the size of the dataset is large and that the prediction function $h(x_i,\theta)$ is a non-linear, non-convex, smooth function. There are two types of numerical optimization algorithms applied to such objective functions: stochastic and batch optimization algorithms. The difference between them can be shown on the simplest first-order optimization method that is called gradient descent (GD) and its stochastic modification stochastic gradient descent (SGD).

1.2.1 Deterministic Optimization

GD is a batch method which uses the whole dataset in every update iteration. The update of the GD is

$$\theta_{t+1} = \theta_t - \alpha\nabla\mathcal{L}(\theta_t). \tag{1.5}$$

As one needs to calculate the whole gradient in each iteration, this process is computationally inefficient for large datasets. One of the ways to deal with this problem is to diminish the number of iterations that are needed to converge. To do this one can use second-order algorithms, that estimate and use the information about the curvature of the loss function to converge faster. There are many second-order methods that approximate the Hessian of the loss function such as Hessian-Free, Quasi-Newton and Natural Gradient methods. Chapter 5 of Bottou, Curtis, and Nocedal, 2018 provides a detailed review of the second-order methods in large-scale machine learning.

1.2.2 Stochastic Optimization

The second type of optimization algorithms uses stochastic estimators of the full loss function and its gradients. For example, the SGD update rule is

$$\theta_{t+1} = \theta_t - \alpha g(\theta_t), \tag{1.6}$$

where $g(\theta)$ is the stochastic gradient that is a stochastic estimator of the full gradient $\nabla\mathcal{L}(\theta_t)$. Stochastic gradient is equal to the gradient of the loss function calculated on a mini-batch $\mathcal{B} \subset \mathcal{D}$,

$$g(\theta) = \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} \nabla l(\theta; x_i). \tag{1.7}$$

It is unbiased estimator of the full gradient

$$\mathbb{E}[g(\theta)] = \nabla\mathcal{L}(\theta). \tag{1.8}$$

As a sum of identically distributed random variables, $g(\theta)$ is normally distributed with

$$g(\theta) \sim \mathcal{N}(\nabla\mathcal{L}(\theta), \operatorname{cov}[g(\theta)]) \tag{1.9}$$

The covariance of the stochastic gradient $\operatorname{cov}[g(\theta)] = \frac{\operatorname{cov}[\nabla l_i(\theta)]}{|\mathcal{B}|}$ is inversely proportional to the batch size $|\mathcal{B}|$. Thus, the mini-batch size determines the trade-off between the cost of one iteration and the variance of the stochastic gradient.

SGD and its modifications are widely used in machine learning. However, the noise in the gradient estimate causes several theoretical and practical problems. In contrast to GD, for which you can prove linear convergence with a small enough step size (Bottou, Curtis, and Nocedal, 2018), SGD has only sub-linear convergence (Bottou, Curtis, and Nocedal, 2018). Moreover, it is possible to show that for any decent direction, the convergence of such stochastic algorithm will still be sub-linear. This problem was partially solved by variance reduction methods such as SAGA(Defazio, Bach, and Lacoste-Julien, 2014), SVRG (Johnson and Zhang, 2013). These methods enjoy linear convergence, however, they have several restrictions on their usage. For instance, SAGA needs a large amount of memory to store previous values of the gradients, whereas SVRG needs to compute the full gradient during their iteration, so their usefulness for large-scale non-convex machine learning still should be proven.

1.3 Specialized Optimization Methods for Deep Learning

There are several methods that proved to be effective in the training of neural networks. Surprisingly, SGD and its modifications such as SGD with Momentum have been shown to be really effective when they are applied with carefully tuned learning rate schedules. Another direction of work is based on the variance adaptation. Such optimizers as RMSprop (Tieleman and Hinton., 2012), ADAM (Kingma and Ba, 2015) or SVAG (Balles and Hennig, 2018) are commonly used during learning of neural network models.

Below, we describe the update rules of the two mostly used by the deep learning community optimization methods.

1.3.1 SGD with Momentum

Despite the fact that neural networks are non-convex, non-linear functions, a simple modification of SGD, SGD with Momentum (Polyak, 1964), with careful tuning of hyperparameters, gives state-of-the-art results on different neural network architectures such as ResNets (He et al., 2016). For finding an update direction, a moving average of the stochastic gradient is calculated

$$m_t = \mu m_{t-1} + (1-\mu)g_t, \tag{1.10}$$

with $\mu \in (0,1)$.

The update rule for SGD with Momentum is

$$\theta_{t+1} = \theta_t - \alpha_t m_t. \tag{1.11}$$

For detailed description of the Momentum and the reasons of its effectiveness we refer the reader to Goh, 2017.

1.3.2 ADAM

ADAM (Kingma and Ba, 2015) is one of the most used optimizes for deep learning applications. It maintains moving averages of stochastic gradients and their element-wise square,

$$\tilde{m}_t = \beta_1 \tilde{m}_{t-1} + (1-\beta_1)g_t, \qquad m_t = \frac{\tilde{m}_t}{1-\beta_1^{t+1}}, \tag{1.12}$$

$$\tilde{v}_t = \beta_2 \tilde{v}_{t-1} + (1-\beta_2)g_t^2, \qquad v_t = \frac{\tilde{v}_t}{1-\beta_2^{t+1}}, \tag{1.13}$$

with $\beta_1, \beta_2 \in (0,1)$. Then, bias-correction is applied to moving averages. It is needed because of the zero initialization of stochastic gradient. Finally, the fraction of the first and the second moment is used to update parameters.

The update rule for ADAM is

$$\theta_{t+1} = \theta_t - \alpha \frac{m_t}{\sqrt{v_t} + \varepsilon}. \tag{1.14}$$

1.4 Step Size Optimization

An additional important problem related to the stochastic optimization framework is the choice of a step size α in the stochastic case. The step size is one of the most important hyperparameters of machine learning models that dramatically influences the learning process (Bengio, 2012). Our objective is to find a step size α or, more general, a step size schedule $\{\alpha_t\}_{t=1}^{T-1}$ such that the expected risk $R(\theta_T(\{\alpha_t\}_{t=1}^{T-1})$ is minimal. As a good approximation of the expected risk, one can use cross-validation loss. We can tune the step size schedule to find the minimum of the cross-validation loss. Tuning of the step size schedule is challenging because the gradients of the cross-validation loss function are usually unavailable. There are several ways to deal with this problem. The first solution consists of the use of the zero-order optimization of the cross-validation loss as a function of step size which does not need gradients of the loss function. The methods such as a random search (Bergstra and Bengio, 2012) or Bayesian optimization (Snoek, Larochelle, and Adams, 2012) were successfully applied to such hyperparameters tuning including step size hyperparameter.

The most common way to choose the step size is a manual search over a list with reasonable step sizes. In this case, one trains the model with the predefined step size or the step size schedule and, after the whole training process, one looks at the cross-validated performance of the model. The values that give the best performance on the validation dataset can be chosen by doing the whole optimization with several step sizes or step size schedules. This method requires quite a lot of computational resources because one should do the whole optimization of the parameters for every step size.

Another solution could be described as reversible learning where gradients of the cross-validation loss with respect to step size schedule were propagated back through the entire training procedure. Recently, Maclaurin, Duvenaud, and Adams, 2015 computed exact gradients of cross-validation performance with respect to all hyperparameters by chaining derivatives backwards through the entire training procedure. Reversible learning is not only one attempt to use first-order information for optimization of the hyperparameters. For example, Wu et al., 2017 incorporate gradients in the Bayesian Optimization framework.

All the methods previously discussed require calculation of the cross-validation loss after a significant part of training and, as a result, they all are computationally expensive. In contrast to these methods, one can treat the choice of step size as an additional optimization problem of the regularized objective function $\mathcal{L}(\theta)$. Such methods look for a step size that minimizes the regularized loss function for the current iteration.

In the deterministic case, we can choose the optimal step size using line search algorithms such as inexact line search by Wolfe, 1969. However, in the stochastic case, it is not possible to use line search algorithms as they are sensitive to the noise in stochastic gradients. Recently, a modification of this algorithm in the stochastic case, probabilistic line search (Mahsereci and Hennig, 2015), was developed. It uses a Gaussian process to estimate the loss function and makes probabilistic decisions about the optimal step size. Modelling of the loss function with a Gaussian process can still require a lot of computations of the loss function, so one should compare the computational complexity of this method with the computational complexity of previous methods.

Another way to find the step size that minimizes the regularized loss function is to apply an iterative algorithm that adapts the step size in every iteration using an available local information about the loss function and its gradients to make it closer to the optimal step size. We refer to such methods as online step size adaptation methods and discuss them in detail in Chapter 2 as they are the main focus of this thesis.

1.5 Overview

The rest of this thesis is organized as follows. Chapter 2 discusses methods that were used for online adaptation of step size. Two of them, hypergradient descent and stochastic Barzilai-Borwein, are reviewed in detail. Chapter 3 covers some aspects of proximal algorithms that are important for understanding the adaptation rule that we propose. In Chapter 4, we develop the framework that splits the procedure of finding the update rule for adapting step size on three independent components. These components are fitting of some model to the values of the loss function and its gradients, computing of the proximal operator of fitted model and applying one iteration of the proximal point method to adapt step size. Subsequently, we apply this framework to develop the proximal quadratic model for step size adaptation (Chapter 5). We analyze the trade-off between systematic mistakes caused by the regularization and bias of the model because of noise in stochastic gradients. Chapter 6 discusses the numerical experiments and their results.

2 Online Adaptation of Step Size

As finding the optimal step size in the stochastic case can be expensive there are several attempts to use an iterative process that adapts the step size from iteration to iteration, making it closer to the optimal step size (Baydin et al., 2018; Tan et al., 2016; Barzilai and Borwein, 1988; Almeida et al., 1999; Schraudolph, 1999; Schaul, Zhang, and LeCun, 2013). Such updates usually do not require additional computations of the loss function and its gradients. Below we provide a more detailed description of several adaptation algorithms that are most related to our model.

2.1 Hypergradient Descent (HD) Adaptation

Hypergradient descent (Baydin et al., 2018; Almeida et al., 1999) is an effective and simple method for online step sizes adaptation. The main idea behind its update rule is that we can use gradient descent to find the optimal step sizes for our optimization algorithm. Most of the update rules can be described as the linear function of step size α_t

$$\theta_{t+1}(\alpha_t) = \theta_t + \alpha_t \upsilon_t, \tag{2.1}$$

where υ_t is an update direction and θ_t is current value of the parameters. The optimal step size for such update could be described as

$$\alpha_t^* = \arg\min_{\alpha_t} \mathcal{L}(\theta_{t+1}(\alpha_t)). \tag{2.2}$$

To find the optimal step size α_t^* we can use standard iterative algorithms such as GD. The majority of these algorithms need at least the first order information, e.g. gradients of the loss function as a function of step size $\frac{\partial \mathcal{L}(\theta_{t+1}(\alpha_t))}{\partial \alpha_t}$. Using the chain rule we have

$$\frac{\partial \mathcal{L}(\theta_{t+1}(\alpha_t))}{\partial \alpha} = \nabla_\theta \mathcal{L}(\theta_{t+1}) \frac{\partial \theta_{t+1}}{\partial \alpha} = \nabla_\theta \mathcal{L}(\theta_{t+1})^T \upsilon_t. \tag{2.3}$$

We will refer to this gradient as the hypergradient.

As we do not have access to the full loss we will use the unbiased estimate of hypergradient. Under the assumption that the noise at step $t+1$ is independent from the noise at previous iterations, it is equal to the scalar product of the stochastic gradient g_{t+1} and the current step υ_t

$$\mathbb{E}[g_{t+1}^T \upsilon_t] = \mathbb{E}[g_{t+1}]^T \upsilon_t = \nabla_\theta \mathcal{L}(\theta_{t+1})^T \upsilon_t. \tag{2.4}$$

So we can write down the stochastic HD update rule:

$$\alpha_t = \alpha_{t-1} - \beta g_{t+1}^T \upsilon_t. \tag{2.5}$$

However, we cannot compute g_{t+1} without α_t. If we assume that the optimum value of the step size at each iteration does not change much, we can use the step size update from previous step in the current iteration

$$\alpha_t = \alpha_{t-1} - \beta g_t^T \upsilon_{t-1}. \tag{2.6}$$

We implemented this algorithm for SGD with Momentum. Pseudocode is presented in the Algorithm 1.

Algorithm 1 Momentum with HD adaptation

```
Require: initial parameter value $\theta_0$, initial step size $\alpha_0$, Hypergradient step size
    $\beta$,momentum $\mu$, number of steps $T$
1: Initialize $\upsilon = 0$, $m = 0$, $\alpha = \alpha_0$
2: for $t = 1, \ldots, T$ do
3:     Evaluate stochastic gradient $g$
4:     Evaluate hypergradient $h = g^T \upsilon$
5:     Adapt step size $\alpha = \alpha - \beta h$
6:     Update moving average $m = \mu m + (1 - \mu)$
7:     Evaluate new direction $\upsilon = -m$
8:     Update parameters $\theta = \theta + \alpha \upsilon$
9: end for
```

2.2 Stochatic Barzilai-Borwein Step Size

The Barzilai and Borwein, 1988 method is motivated by quasi-Newton methods (Boyd and Vandenberghe, 2004). A typical iteration of quasi-Newton methods for solving Equation (1.4) is

$$\theta_{t+1} = \theta_t - B_t^{-1} \nabla_\theta \mathcal{L}(\theta_t), \tag{2.7}$$

where B_t is an approximation of the Hessian matrix of $\mathcal{L}(\theta)$ at the current iterate θ_t. One of the fundamental feature of the B_t is that it must satisfy the secant equation:

$$B_t s_t = y_t, \tag{2.8}$$

where $s_t = \theta_t - \theta_{t-1} = \alpha_{t-1}\upsilon_{t-1}$ and $y_t = \nabla\mathcal{L}(\theta_t) - \nabla\mathcal{L}(\theta_{t-1})$. The secant equation follows from applying the secant method of finding function roots to the derivative of the second-order approximation of the loss function at the point θ_{t-1}. BB method uses the simple B_t matrix, that is equal to the $\frac{1}{\alpha_t} I$ and tries to solve the Equation (2.8) approximately by minimizing euclidean norm of the difference

$$\min_{\alpha_t} \left\| \frac{1}{\alpha_t} s_t - y_t \right\|_2^2 . \tag{2.9}$$

The solution of this minimization problem gives

$$\alpha_t = \frac{\|s_t\|_2^2}{s_t^T y_t} = \frac{\alpha_{t-1} \|v_{t-1}\|_2^2}{\nabla_\theta \mathcal{L}(\theta_t)^T v_{t-1} - \nabla_\theta \mathcal{L}(\theta_{t-1})^T v_{t-1}} . \tag{2.10}$$

Because the BB algorithm requires computation to the full gradients, it has been inapplicable in the stochastic case for a long time. Recently this algorithm was modified for the stochastic case by Tan et al., 2016. This adaptation requires a large number of parameter updates (inner loop of the optimization step) between two updates of step size (outer loop of the optimization step). This is necessary to get the estimator of the ratio in Equation (2.10) with a small bias. However, the connection between the inferred step size of the outer loop of the optimization and the optimal step size for the inner loop is not clear.

Algorithm 2 Momentum-BB

```
Require: initial parameter value $\theta_0$, initial step sizes $\alpha_0$, momentum $\mu$, number of
    epochs $N$, number of itterations in the one epoch $M$
1:  Initialize $v = 0$
2:  for $n = 0, \dots, N$ do
3:      if $n > 0$ then
4:          Update $\alpha_n = \frac{1}{m} \left\| \tilde{\theta}_n - \tilde{\theta}_{n-1} \right\|_2^2 \backslash (\tilde{\theta}_n - \tilde{\theta}_{n-1})^T (\hat{g}_n - \hat{g}_{n-1})$
5:          $\theta_0 = \tilde{\theta}_n$
6:          $\hat{g}_{n+1} = 0$
7:          for $m = 0, \dots, M-1$ do
8:              Evaluate stochastic gradient $g_m$
9:              Update estimate of the gradient $\hat{g}_{n+1} = \mu \hat{g}_{n+1} + (1-\mu) g_m$
10:             Evaluate new direction $v = \mu v + (1-\mu)(-g_m)$
11:             Update parameters $\theta_{m+1} = \theta_m + \alpha_n v$
12:         end for
13:         $\tilde{\theta}_{n+1} = \theta_m$
14:
```

Both algorithms discussed above try to model the optimal step size without modeling the loss function explicitly. In the case when the full loss function can be computed efficiently, this approach could be sufficient. However, in the stochastic case it could lead to the large bias and inefficient usage of available information. Thus, in the Section 4, we propose the model based approach that allows us to incorporate uncertainty about our observation of stochastic loss and its gradients. For developing the adaptation rule from the fitted model we use an iteration of the proximal point algorithm. The next chapter gives a short introduction to the proximal point algorithms that is sufficient to understand our framework for adaptation of the step sizes.

3 Proximal Point Algorithms

This section provides information about the proximal operator and its usage in proximal point algorithms. The connection of the proximal operator with gradient descent and trust region optimization is discussed. Next, the proximal point algorithm for a quadratic function is described as its iteration was used in Chapter 5. Finally, we discuss recent results in stochastic proximal point algorithms. For a more detailed discussion of the proximal algorithms, the reader is refereed to Parikh and Boyd, 2014.

3.1 Proximal Operator

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a closed proper convex function, then *a proximal operator* $\mathrm{prox}_f : \mathbb{R}^n \to \mathbb{R}^n$ of f is defined by

$$\mathrm{prox}_f(v) = \arg\min_x \left(f(x) + \frac{1}{2} \|x - v\|_2^2 \right). \tag{3.1}$$

The minimized function has a unique minimizer for every $v \in \mathbb{R}^n$. The minimum of the original function could be found by looking at the fixed points of the proximal operator. In other words, $\mathrm{prox}_{\alpha f}(x^*) = x^*$ if and only if x^* minimizes f.

One important application of proximal operators is a proximal operator of the function approximation. For f that is twice differentiable at v, its first-order approximation near v is equal to

$$\hat{f}_v^{(1)}(x) = f(v) + \nabla f(v)^T (x - v) \tag{3.2}$$

and the second-order approximation is

$$\hat{f}_v^{(2)}(x) = f(v) + \nabla f(v)^T (x - v) + \frac{1}{2}(x - v)^T \nabla^2 f(v)(x - v). \tag{3.3}$$

The proximal operators for these approximations are well-known update steps. The proximal operator of the first-order approximation is

$$\mathrm{prox}_{\alpha \hat{f}^{(1)}}(v) = v - \alpha \nabla f(v), \tag{3.4}$$

which is a standard gradient step with step size α. The proximal operator of the second-order approximation is

$$\mathrm{prox}_{\alpha \hat{f}^{(2)}}(v) = v - \left(\nabla^2 f(v) + \frac{1}{\alpha} I \right)^{-1} \nabla f(v), \tag{3.5}$$

which is Tikhonov-regularized Newton's update (Levenberg-Marquart). Application of the proximal operator to the convex approximation of the function can lead to well-known updates that are provably converging to the minimum of the original function. However, there is no general result that links the quality of approximation on each iteration and the size of the trust region α with the convergence of the iterative procedure.

3.2 Trust Region Optimization

The proximal operator of a scaled function αf could be interpreted as an operator that returns a minimizer of the original function with additional constraints. *A trust region* optimization problem is written as

$$\begin{aligned} &\min_x f(x) \\ &\text{subject to } \|x - v\| \le \rho. \end{aligned} \tag{3.6}$$

Such problems typically arise when $f(x)$ is approximation of some function $g(x)$ that is accurate only in some region near the point v (like $\hat{f}_v^{(1)}(x)$ and $\hat{f}_v^{(2)}(x)$). In this case, this function should be minimized inside the region where this approximation is accurate enough. The proximal problem for the scaled function f is

$$\min_x \left(f(x) + \frac{1}{2\alpha} \|x - v\|_2^2 \right). \tag{3.7}$$

The solutions to the proximal problem class are included in the solutions to the trust region problem class. In other words, for every solution of the proximal problem with a scale parameter α, there is a trust region problem with a parameter ρ, such that its solution is equal to the solution of the proximal problem. More specifically, the set of the solutions to the trust region problem class that are located on the border of the trust region is equivalent to the set of solutions of the proximal problem class (Parikh and Boyd, 2014).

3.3 Proximal Point Algorithms

As we discuss above, we can find the minimum of the function f by finding the fixed points of the proximity operator, so such points that

$$x^* = \text{prox}_{\alpha f}(x^*). \tag{3.8}$$

To find such point, iterative process is constructed

$$x_{k+1} = \text{prox}_{\alpha f}(x_k). \tag{3.9}$$

Such process converges to the fixed point of the proximal operator of the original function and, thus, to one of the minimizers of this function.

A special case of proximal point algorithm, iterative refinement (Golub and Wilkinson, 1966), is a proximal point algorithm for the quadratic function

$$f(x) = \frac{1}{2}x^T Ax + b^T x + c. \tag{3.10}$$

For the quadratic function it is possible to find the closed form solution of the proximal problem. The update rule of iterative refinement iterative process is

$$x^{k+1} = \left(A + \frac{1}{\alpha}I\right)^{-1} \left(\frac{1}{\alpha}x^k - b\right) \tag{3.11}$$

We would use this iteration in one-dimensional case to construct the adaptation based on a quadratic model of the loss function.

3.4 Stochastic Proximal Point Algorithms

In the situations when it is inefficient to use GD because of the size of data set, SGD is often used. SGD uses just gradient information but not the entire function and this seems sub-optimal in situations when the proximal operator of the function could be computed easily. In these situations, the stochastic proximal point (SPP) (Patrascu and Necoara, 2017) algorithm is a good alternative to SGD. Recent theoretical results Patrascu and Necoara, 2017 show a non-asymptotic convergence of the stochastic proximal point algorithm.

4 Proximal Step Size Adaptation

We propose a framework to develop an update rule for online adaptation of step size. This framework consists of three parts. First, we approximate a one-dimensional projection of the loss function to the current update direction by a convex function. In the stochastic case, we do not have access to the exact values of the function and its gradients so we need to fit the model with their stochastic estimators. Secondly, we calculate the proximal operator of the fitted model. This could be done analytically or using several iterations of the classical Newton's method. Finally, we construct the adaptation rule as one step of the proximal point algorithm.

4.1 Modeling of the Loss Function

The usual choice of the local approximation is a linear or quadratic model. However, some functions are not well-approximated by a linear or quadratic model, leading to slow convergence. Thus, for such functions a more appropriate choice of the local model for loss function such as Cauchy approximation was proposed (Minka, 2000). The choice of the model depends on the available information. In case of approximation of one-dimensional loss function as a function of the step size, one can compute the values of this function and its gradients at several points just by reusing values of the gradient computed during the previous step. Usage of this information allows us to fit our model without the cost of additionally computing the loss function and its gradient in new points.

The loss function $\mathcal{L}(\theta_t + \alpha \upsilon_t)$ could be treated as a function of step size α when the update direction υ_t is fixed. Let us use $f_t(\alpha) = \mathcal{L}(\theta_t + \alpha \upsilon_t)$ to denote this function. We can approximate this function by some convex function $h_t(\alpha)$. There are many possible variants of the loss function approximation depending on the available information. For example, if only local values of the gradients are given at some point, the best model would be just a linear model (such as Equation (4.4)). The derivative of the function $f_t(\alpha)$ with respect to α at α_t is equal to the hypergradient

$$f_t'(\alpha_t) = \nabla_\theta \mathcal{L}(\theta_{t+1})^T \upsilon_t. \tag{4.1}$$

In other words, HD adaptation uses information about the derivative of the function $f_t(\alpha)$ at the previous step size value to adapt its value. However, additional information about $f_t(\alpha)$ is available, so it is possible to construct a more accurate approximation of the loss function.

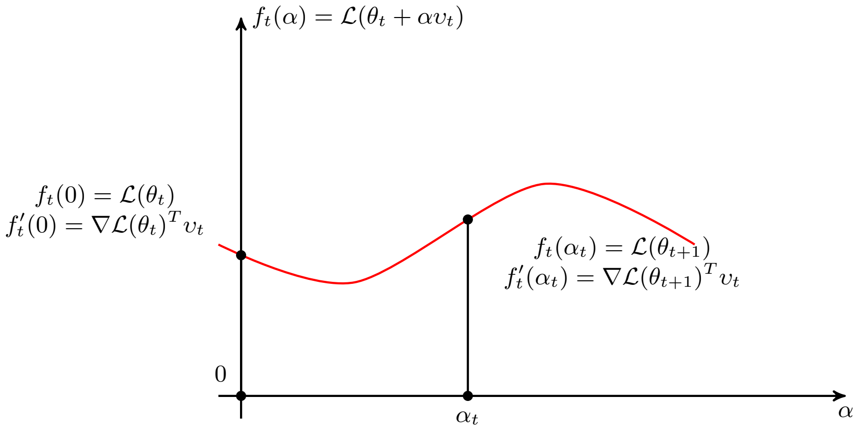

$f_t(\alpha) = \mathcal{L}(\theta_t + \alpha \upsilon_t)$
$f_t(0) = \mathcal{L}(\theta_t)$
$f_t'(0) = \nabla\mathcal{L}(\theta_t)^T\upsilon_t$
$f_t(\alpha_t) = \mathcal{L}(\theta_{t+1})$
$f_t'(\alpha_t) = \nabla\mathcal{L}(\theta_{t+1})^T\upsilon_t$
0
α_t
α

FIGURE 4.1: Illustration of the available information that can be used for the adaptation of step size. We can compute $f_t(0)$ and $f_t'(0)$ during current iteration t and, additionally, we can compute $f_t(\alpha)$ and $f_t'(\alpha)$ during next iteration $t+1$.

We can easily get access to the derivatives and values of $f_t(\alpha)$ at point $\alpha = 0$ by saving the previous value $f_t(0) = f_{t-1}(\alpha_{t-1})$ and computing the scalar product of the previous gradient $\nabla_\theta\mathcal{L}(\theta_t)$ and the current update direction υ_t

$$f_t'(0) = \nabla_\theta\mathcal{L}(\theta_t)^T\upsilon_t. \tag{4.2}$$

Thus, even without significant additional memory and computational resources(e.g. by computing one scalar product and saving two one-dimensional values) the values $f_t(0), f_t(\alpha), f_t'(0), f_t'(\alpha)$ are accessible. Figure 4.2 depicts an illustration of the function $f_t(\alpha)$ and the information that can be used for the estimation of this function and its minimum. In the stochastic case, the exact function values and its derivatives are usually not computed but we could use their stochastic estimators instead. In such a situation, the model is usually fitted to maximize the likelihood of the given estimators.

4.1.1 Restriction on the Model Usage

It is important to apply the adaptation of the step size only in cases when it is possible to fit the convex model from a chosen set of functions (e.g. parametric family of quadratic models). There are situations when maximum likelihood parameters for the optimal model are such that our model $h_t(\alpha)$ is not convex (e.g. in case of the quadratic model described in Equation (5.1) this happens when $w_2 < 0$). The adaptation of the step size is not possible using the fitted model. There are several possible solutions to this problem. One solution would be to approximate the loss function with another model. Another, even simpler solution, would be not to update the step size at all (this could be treated as applying the simplest constant model).

4.2 Finding the Proximal Operator of the Model

After fitting the model to the available information about the function, one has the access to the convex model of the one-dimensional projection of the loss function to the current update direction. As a convex function, this model has a unique minimum, that could be used for the adaptation of the step size. However, its value could be far from the minimum of the underlying loss function, because of the noise in stochastic gradients and an error in approximation by a convex function. So to make sure that the adaptation would not change the step size significantly, we can look for the minimum of the model in the trust region near the previous step size α_t.

Following the results presented in Section 3.2, the trust region optimization problem could be substituted by the proximal problem as it is easier to solve in many cases. For some models such as linear and quadratic models, this operator could be computed in the closed form. However, it is cheap to compute it for other convex functions because the problem is one-dimensional. For example, one can use several iterations of the classical Newton's method.

4.3 Proximal Point Iteration for Step Size Adaptation

We would like to develop the update rule for step size adaptation. For this, we can use one iteration of the proximal point algorithm applied to the fitted model $\hat{h}_t(\alpha)$. We propose to use

$$\alpha_{t+1} = \operatorname{prox}_{\beta \hat{h}_t}(\alpha_t) \tag{4.3}$$

as the step size update for the optimization algorithm. Such adaptation combines the information about the minimum of the model with the previous optimal step size value. A parameter β corresponds to the size of the trust region in which our stochastic approximation $\hat{h}_t(\alpha)$ is still accurate enough. As the parameter β increases, we rely more on the optimum of the fitted model $\hat{h}_t(\alpha)$. Unfortunately, we cannot determine the optimal size of the trust region, so we still should tune this parameter β. However, given that the model $h_t(\alpha)$ has enough capacity and the noise in the measurements of the loss function is small enough the sensitivity to this parameter is smaller and larger β values could be used to adapt the step size faster. In the next chapters, we will show how the change from a linear model to quadratic affects the sensitivity to this hyperparameter and makes it's tuning easier. For an illustration of usage of this framework, we could derive the HD update rule using this framework.

4.4 HD as Proximal Point of the First-order Approximation of the Loss Function

The function $f_t(\alpha)$ can be approximated by the linear function $h_{lin}(\alpha)$ that is going through $f_t(\alpha_t)$ with the scope that is equal to hypergradient $f_t'(\alpha_t)$,

$$h_{lin}(\alpha) = f_t'(\alpha_t) \cdot (\alpha - \alpha_t) + f_t(\alpha_t). \tag{4.4}$$

The next step is to find the proximal operator of the function $h_{lin}(\alpha)$ in the point α_t. For the linear model, this could be done in the closed form. The proximal operator is

$$\operatorname{prox}_{\beta h_{lin}}(\alpha_t) = \arg\min_{\alpha} h_{lin}(\alpha) + \frac{1}{2\beta}(\alpha - \alpha_t)^2 \tag{4.5}$$

Applying one iteration of proximal point algorithm

$$\alpha_{t+1} = \operatorname{prox}_{\beta h_{lin}}(\alpha_t), \tag{4.6}$$

we obtain exactly HD update

$$\alpha_{t+1} = \alpha_t - \beta f_t'(\alpha_t). \tag{4.7}$$

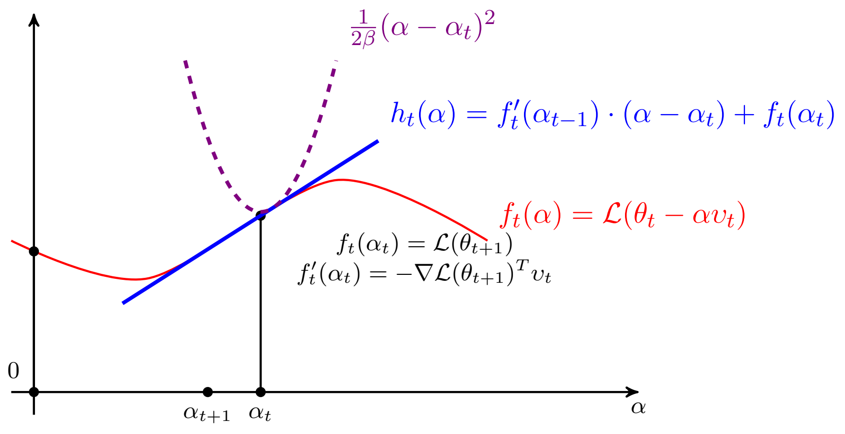

$\frac{1}{2\beta}(\alpha - \alpha_t)^2$
$h_t(\alpha) = f_t'(\alpha_{t-1}) \cdot (\alpha - \alpha_t) + f_t(\alpha_t)$
$f_t(\alpha) = \mathcal{L}(\theta_t - \alpha v_t)$
$f_t(\alpha_t) = \mathcal{L}(\theta_{t+1})$
$f_t'(\alpha_t) = -\nabla\mathcal{L}(\theta_{t+1})^T v_t$
0
α_{t+1}
α_t
α

Figure 4.2: Hypergradient Descent adaptation as an iteration of proximal point algorithm applied to the linear model. We use the local information $f_t(\alpha_t)$ and $f_t'(\alpha_t)$ (or their stochastic estimators) to fit the linear model $h_t(\alpha)$ (blue line) to the loss function $f_t(\alpha)$ (red line). The next α_{t+1} is equal to the value of the proximal operator of the $h_t(\alpha)$ in the point α_t. To find it we should minimize the sum of our model $h_t(\alpha)$ and the regularization term $\frac{1}{2\beta}(\alpha - \alpha_t)$.

Therefore, it is possible to interpret a recently rediscovered HD adaptation rule as a combination of the linear model of the loss function and proximal point iteration for this model. However, following the same framework it is possible to develop other update rules that are less sensitive to the hyperparameters choice and can show better performance. For this we apply our framework to the quadratic model that was fitted to the stochastic estimators of function values and gradients.

5 Proximal Quadratic (PQ) Adaptation

In this section, we investigate the properties of the quadratic model for step size adaptation and develop a proximal quadratic adaptation for the step size. Following the framework that was described in Chapter 4 we develop PQ step size adaptation in three steps. The first part is modelling and fitting of the loss function by a quadratic function. The second part is finding a proximal operator of the quadratic function in closed form. The last part consists of the application of the update rule in the case when it is possible to fit a convex quadratic model to the given observations.

5.1 Fitting the Quadratic Model

Following the interpretation of the HD adaptation as a linear model with quadratic regularization, we propose to use a quadratic model as the model for the loss function $\mathcal{L}(\theta_{t+1}(\alpha))$ as the function of step size α. The quadratic model for the loss function $\mathcal{L}(\theta_{t+1}(\alpha))$ could be written as

$$h_{quad}(\alpha) = w_2\alpha^2 + w_1\alpha + w_0, \tag{5.1}$$

where $\{w_i\}_{i=0}^2$ are the parameters of the quadratic model $h_{quad}(\alpha)$.[1] We estimate these parameters using all available information about the loss function. This information is the stochastic estimates of the loss function and its derivative. When we are using mini-batches to estimate the loss function and its gradient these estimates are random variables that are approximately normally distributed as a sum of the i.i.d. random variables. Fitting of the model consists of finding the maximum likelihood solution in the class of quadratic functions with additive normal noise given stochastic estimates $\hat{f}, \hat{f}(\alpha_t), \hat{f}'(0), \hat{f}'(\alpha_t)$ of $f(0), f(\alpha_t), f'(0), f'(\alpha_t)$ and the variance of these estimates.

This problem can be formulated as linear regression

$$y = Xw + \varepsilon, \tag{5.2}$$

where w is vector of coefficients of the quadratic model and ε_i is normally distributed additive noise $\varepsilon_i \sim \mathcal{N}(0, \sigma_i^2)$. The vector y and the matrix X are constructed in the way to fit a second-order polynomial to the estimators of the function values at two

[1]Here and later in this chapter, we drop the index t for function $f_t(\alpha)$ and its model $h_t(\alpha)$ using just $f(\alpha)$ and $h(\alpha)$ for better readability.

points α_t and 0 and the estimators of the function derivatives at these points

$$y = \begin{bmatrix} \hat{f}(\alpha_t) \\ \hat{f}(0) \\ \hat{f}'(\alpha_t) \\ \hat{f}'(0) \end{bmatrix}, X = \begin{bmatrix} 1 & \alpha_t & \alpha_t^2 \\ 1 & 0 & 0 \\ 0 & 1 & 2\alpha_t \\ 0 & 1 & 0 \end{bmatrix}. \tag{5.3}$$

To find the optimal parameters we minimize the negative loglikelihood:

$$-LL(w) = const(w) + \sum_{i=1}^{4} \frac{(y_i - x_i^T w)^2}{2\sigma_i^2} = const(w) + const(w) \cdot \sum_{i=1}^{4} \left(\frac{y_i}{\sigma_i} - \left(\frac{x_i}{\sigma_i}\right)^T w \right)^2. \tag{5.4}$$

Finding the maximum likelihood solution for a linear model with different noise is equivalent to the minimum a least squares problem with scaled inputs and outputs:

$$\min_w ||\tilde{y} - \tilde{X}w||_2^2, \tag{5.5}$$

where elements of $\tilde{y}$ and $\tilde{X}$ are $\tilde{y}_i = \frac{y_i}{\sigma_i}, \tilde{x}_{ij} = \frac{x_{ij}}{\sigma_i}$. The solution to this optimization problem is

$$\hat{w} = \left(\tilde{X}^T \tilde{X}\right)^{-1} \tilde{X}\tilde{y}. \tag{5.6}$$

The full expression for maximum likelihood parameters $\hat{w}$ in the general case is presented in the first part of the Appendix. Here, we use several assumptions that simplify the expression (A.1) for estimated parameters to a much simpler form that does not contain the variances σ_i^2.

Assumption 1. *The variance of the stochastic loss and gradient measurements is approximately equal in different points. So using the notation from Equation* (5.2)*, we have* $\sigma_1 \approx \sigma_2 = \sigma_{loss}$ *and* $\sigma_3 \approx \sigma_4 = \sigma_{grad}$.

Assumption 2. *The variance of the stochastic loss is much larger than the variance of the stochastic estimators of function derivatives:* $\sigma_{loss} \gg \sigma_{grad}$.

Under the Assumptions 1 and 2 the second and third component of the estimated parameters $\hat{w}$ would simplify to

$$\hat{w}^1 \approx \hat{f}'(0), \qquad \hat{w}^2 \approx \frac{\hat{f}'(\alpha_t) - \hat{f}'(0)}{2\alpha_t}. \tag{5.7}$$

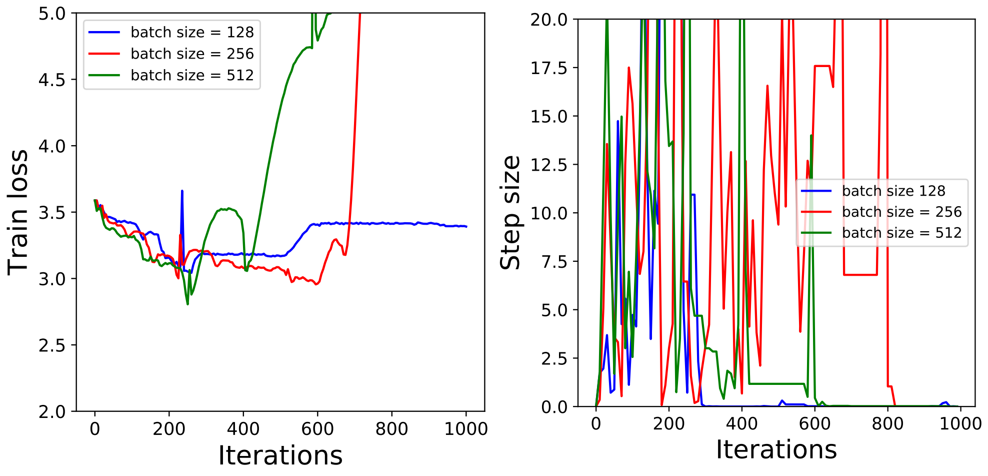
5.2 Optimum of the Quadratic Model

After the parameters of the quadratic model are estimated, we can find the minimum of the model with the estimated parameters. If the model fits the data well we could expect that the minimum value of the model is close to the minimum of the original model.

The minimum of the quadratic model $h_{quad}(\alpha)$ defined in Equation (5.1) is equal to $\alpha_{opt} = -\frac{w_1}{2w_2}$ for $h''_{quad}(\alpha) = 2w_2 > 0$. Using the maximum likelihood estimators of the parameters under Assumptions 1 and 2, we obtain the estimator for the minimum of the quadratic model

$$\hat{\alpha}_{opt} \approx -\frac{\hat{f}'(0)}{\hat{f}'(\alpha) - \hat{f}'(0)}\alpha_t. \tag{5.8}$$

One can try to use the minimum of the quadratic model to adapt the step size. However, such an adaptation rule is quite unstable. In Figure 5.1, we show that an implementation of such adaptation algorithm (Pseudocode for this algorithm is presented in part C of the Appendix as Algorithm 4) diverges even for large batch sizes during the first thousand optimization iterations.



(A) Training loss for first 1000 iterations. (B) Step size for first 1000 iterations.

FIGURE 5.1: Divergence of the quadratic model for different batch sizes. Step sizes are unstable and much larger than optimal step size. The experiment was done using SVHN dataset (P2).

In the next section, we show that such an adaptation rule is biased for large step sizes when the noise in the gradient estimates is large or the derivatives of the loss function have similar values.

5.3 Bias of the Quadratic Model

The stochastic estimates $\hat{f}'(0)$ and $\hat{f}'(\alpha_t)$ can be calculated as $\hat{f}'(0) = g(\theta_{t-1})^T v_{t-1}$ and $\hat{f}'(\alpha_t) = g(\theta_t)^T v_{t-1}$. They are unbiased estimators of $f'(0)$ and $f'(\alpha_t)$ with

$$\hat{f}'(0) \sim \mathcal{N}(f'(0), \sigma_0^2)$$

$$\hat{f}'(\alpha) \sim \mathcal{N}(f'(\alpha), \sigma_\alpha^2),$$

where $f'(0) = \nabla\mathcal{L}(\theta_{t-1})^T v_{t-1}$ and $f'(\alpha) = \nabla\mathcal{L}(\theta_t)^T v_{t-1}$.
However, the optimal step size suggestion α_{opt} is biased

$$\mathbb{E}[\hat{\alpha}_{opt}] = \mathbb{E}\left[-\frac{\hat{f}'(0)}{\hat{f}'(0) - \hat{f}'(\alpha_t)}\right]\alpha_t \neq -\frac{\mathbb{E}[\hat{f}'(0)]}{\mathbb{E}[\hat{f}'(0)] - \mathbb{E}[\hat{f}'(\alpha_t)]}\alpha_t. \tag{5.9}$$

Using the Taylor expansion, we can get approximation of the expectation of the ratio of two random variables (Stuart and Ord, 1998):

$$\mathbb{E}\left[\frac{X}{Y}\right] \approx \frac{\mu_x}{\mu_y} - \frac{Cov(X,Y)}{\mu_y^2} + \frac{\mathbb{V}\mathrm{ar}[Y]\mu_x}{\mu_y^3}. \tag{5.10}$$

Applying it to the expectation in Equation (5.9) we have

$$\mathbb{E}\left[\frac{\hat{f}'(0)}{\hat{f}'(\alpha) - \hat{f}'(0)}\right] \approx \frac{f'(0)}{f'(\alpha) - f'(0)} + \underbrace{\frac{\sigma_0^2}{(f'(0) - f'(\alpha))^2} + \frac{\left(\sigma_0^2 + \sigma_\alpha^2\right) f'(0)}{(f'(0) - f'(\alpha))^3}}_{bias}. \tag{5.11}$$

Since we need to correct for this bias, in situations where the difference $f'(0) - f'(\alpha)$ is small or the noise in stochastic estimates $\hat{f}'(\alpha)$ or $\hat{f}'(0)$ is large.

A similar problem was discussed in Tan et al., 2016 as their update rule is similar to the quadratic update rule and also biased towards larger values. The authors decided to decrease the noise in the estimators $\hat{f}'(0)$ and $\hat{f}'(\alpha)$ to estimate the ratio in Equation (5.11) more accurately. They made many update iterations (e.g. one epoch) and then add all of them as one iteration to estimate the derivatives $f'(\alpha)$ and $f'(0)$ more accurately. For a detailed discussion of their approach, we refer the reader to Section 2.2.

5.4 Proximal Point Iteration for the Quadratic Model

As we have seen in previous section, the minimum of the quadratic model could not be used as the step size update because such an update is biased toward larger step sizes and thus unstable. One way to make our model more stable is to rely on its prediction only partially and partially stay near the previous point. This could be done by finding the proximal operator of the quadratic model. There is a closed-form solution for the proximal operator of the quadratic function. We discuss its properties and compare them with the properties of the update derived as a minimum of the quadratic model without the proximity term. The proximal operator returns the point that is the minimizer of the sum of the quadratic model and proximity term

$$\mathrm{prox}_{\beta h_{quad}}(\alpha_t) = \arg\min_{\alpha} h_{quad}(\alpha) + \frac{1}{2\beta}(\alpha - \alpha_t)^2. \tag{5.12}$$

To find it we should take the derivative and set it to zero

$$\mathrm{prox}_{\beta h_{quad}}(\alpha_t) = \frac{\frac{1}{\beta}\alpha_t - w_1}{2w_2 + \frac{1}{\beta}}. \tag{5.13}$$

We formulate the update rule as one iteration of the proximal point algorithm. The next step size is

$$\alpha_{t+1} = \frac{\frac{1}{\beta}\alpha_t - w_1}{2w_2 + \frac{1}{\beta}}. \tag{5.14}$$

Using maximum-likelihood estimation $\hat{w}$ the parameters w given in Equation (5.7) we obtain the estimator of the regularized step size update

$$\hat{\alpha}_{t+1} = \frac{\frac{1}{\beta}\alpha_t - \hat{f}'(0)}{\frac{\hat{f}'(\alpha)-\hat{f}'(0)}{\alpha_t} + \frac{1}{\beta}}. \tag{5.15}$$

This step size update is more stable to the noise in stochastic estimates of function

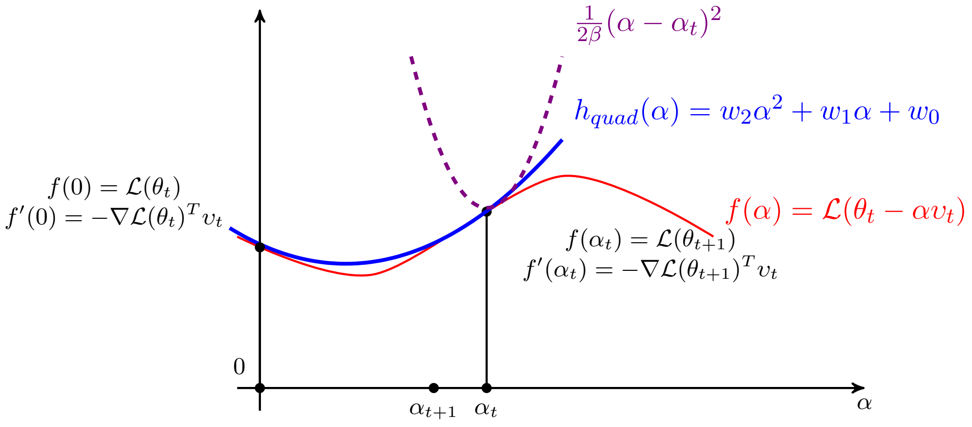

$\frac{1}{2\beta}(\alpha-\alpha_t)^2$
$h_{quad}(\alpha) = w_2\alpha^2 + w_1\alpha + w_0$
$f(0) = \mathcal{L}(\theta_t)$
$f'(0) = -\nabla\mathcal{L}(\theta_t)^T v_t$
$f(\alpha) = \mathcal{L}(\theta_t - \alpha v_t)$
$f(\alpha_t) = \mathcal{L}(\theta_{t+1})$
$f'(\alpha_t) = -\nabla\mathcal{L}(\theta_{t+1})^T v_t$
0
α_{t+1}
α_t
α

FIGURE 5.2: Proximal Quadratic model for step size adaptation. We use function values $f(0)$ and $f(\alpha_t)$ and the derivatives $f'(0)$ and $f'(\alpha_t)$ (or their stochastic estimators) to fit the quadratic model $h_{quad}(\alpha)$ (blue line) to the loss function projection on the current update direction $f(\alpha)$ (red line). The next α_{t+1} is equal to the value of the proximal operator of the $h_{quad}(\alpha)$ in the point α. To find it we should minimize the sum of our model $h_{quad}(\alpha)$ and the regularization term $\frac{1}{2\beta}(\alpha-\alpha_t)$.

derivatives. For small values of the regularization parameter β, the bias of this update is almost equal to zero

$$\mathbb{E}\left[\frac{\frac{1}{\beta}\alpha_t - \hat{f}'(0)}{\frac{\hat{f}'(\alpha)-\hat{f}'(0)}{\alpha_t} + \frac{1}{\beta}}\right] \approx \frac{\frac{1}{\beta}\alpha_t - f'(0)}{\frac{f'(\alpha)-f'(0)}{\alpha_t} + \frac{1}{\beta}}. \tag{5.16}$$

However, the smaller the β the bigger the difference between the optimal quadratic prediction $\frac{f'(0)}{f'(\alpha)-f'(0)}$ and regularized quadratic prediction $\frac{\frac{1}{\beta}\alpha_t - f'(0)}{\frac{f'(\alpha)-f'(0)}{\alpha_t}+\frac{1}{\beta}}$. Thus, the regularization parameter β determines the trade-off between sensitivity to noise and the speed of step size adaptation.

We implement PQ adaptation for SGD with Momentum. Algorithm 3 contains pseudocode for this modification of SGD with Momentum. However, the usage of such adaptation is not restricted to the SGD with Momentum as it is easy to modify other optimization algorithms such as ADAM to use PQ adaptation.

Algorithm 3 Momentum with Proximal Quadratic adaptation (PQ-Momentum)

```
Require: initial parameter value $\theta_0$, initial step size $\alpha_0$, regularization constant $\beta$,
    momentum $\mu$, number of steps $T$, upper bound on Lipschitz constant $M$
 1: Initialize $v = 0$, $m = 0$, $\alpha = \alpha_0$
 2: for $t = 1, \dots, T$ do
 3:     Evaluate stochastic gradient $g$
 4:     Evaluate one-dimentinal derivatives $\hat{f}'(\alpha) = g^T v$ and $\hat{f}'(0) = g_{old}^T v$
 5:     if $0 \le \frac{\hat{f}'(\alpha) - \hat{f}'(0)}{\alpha} \le M$ or $f'(0) > 0$ then
 6:         Update $\alpha = \frac{\frac{1}{\beta}\alpha - \hat{f}'(0)}{\frac{\hat{f}'(\alpha) - \hat{f}'(0)}{\alpha} + \frac{1}{\beta}}$
 7:     end if
 8:     Update moving average $m = \mu m + (1 - \mu)$
 9:     Evaluate new direction $v = -m$
10:     Update parameters $\theta = \theta + \alpha v$
11:     Update $g_{old} = g$
12: end for=0
```

5.5 Quadratic Model Applicability

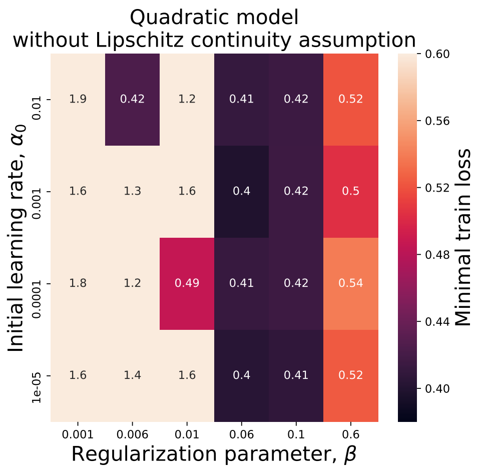
As we discussed in Section 4.1.1, the fitted model could be not convex. For the quadratic model (Equation (5.1)), this happens when the fitted coefficient near the quadratic term $\hat{\theta}_2$ is negative.

Also, as our coefficients are estimated using stochastic estimators of the function derivatives, it is important to make sure that our adaptation of the step size is not dominated by this noise. This could happen in the situation when we estimate the loss in the points that are near to each other. To avoid this we take additional assumption that the derivatives in our model are Lipschitz continuous with constant M

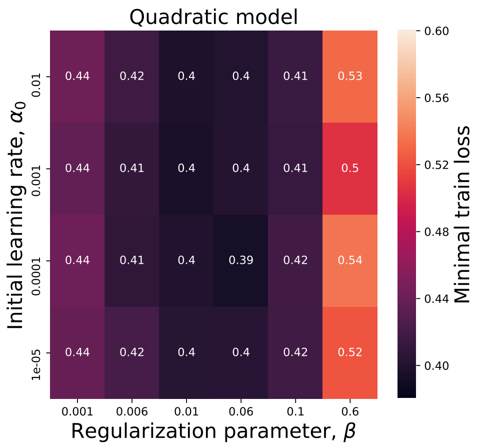
$$\frac{f'(\alpha) - f'(0)}{\alpha} \le M. \tag{5.17}$$

Thus, in the situation when convexity or Lipschitz continuity is violated in the fitted model, we discard the fitted model and use the simplest constant model for the step size. In addition, we don't want to modify the step size in situations when there is no descent or in other words, when $\hat{f}'(0)$ is negative. In such a situation the model would predict a negative step size that is non-optimal globally. We implement this transition between the quadratic and constant model in line 5 of Algorithm 3. There we used an upper bound to Lipschitz (e.g. some large constant such that we are not updating the step size when the computations are not dominated by the noise).

To show the importance of the Lipschitz continuity assumption, we provide the results for the modification of Algorithm 3 without restriction on the $\frac{\hat{f}'(\alpha) - \hat{f}'(0)}{\alpha}$ (Figure 5.3). One can see that without this restriction PQ adaptation fails for small β, whereas for other β it works with this restriction as precise as without it.



α_0 \ β	0.001	0.006	0.01	0.06	0.1	0.6
0.01	1.9	0.42	1.2	0.41	0.42	0.52
0.001	1.6	1.3	1.6	0.4	0.42	0.5
0.0001	1.8	1.2	0.49	0.41	0.42	0.54
1e-05	1.6	1.4	1.6	0.4	0.41	0.52



α_0 \ β	0.001	0.006	0.01	0.06	0.1	0.6
0.01	0.44	0.42	0.4	0.4	0.41	0.53
0.001	0.44	0.41	0.4	0.4	0.41	0.5
0.0001	0.44	0.41	0.4	0.39	0.42	0.54
1e-05	0.44	0.42	0.4	0.4	0.42	0.52

FIGURE 5.3: An effect of Lipschitz continuity restriction. One can see that the model without Lipschitz continuity constrain is not converging to the minimum for small values of β. Minimum loss values of large part of the training dataset. The minimum was chosen from the values loss function computed during fixed number of optimization iterations that is enough for convergence. For both Algorithm 3 and Algorithm 4 the value of momentum parameter is $\mu = 0.99$.

6 Experiments

We experimentally compare PQ with HD adaptation on three problems with different difficulty. First, we compare the algorithms using fitted hyperparameters to show that PQ adaptation finds a better solution given the same computational resources.

Both of these models have two hyperparameters: the initial step size α_0 and the regularization parameter β. As the initial idea of step size adaptation is to make the choice of hyperparameters easier, it is important to compare sensitivity of these hyperparameters. In addition, we also compared the sensitivity of the SGD with Momentum without step size adaptation to the step size to show that both methods are less sensitive to the hyperparameter β than SGD with Momentum to step size α.

6.1 Experimental Set-Up

We tested the optimizers on three problems: CNN training on the MNIST, CIFAR-10 and SVHN datasets. On MNIST, we used CNN with two convolutional layers, interspersed with max-pooling, and two fully-connected layers. On CIFAR-10 and SVHN, we used a similar CNN with three convolutional layers and three fully-connected layers. We used a cross-entropy loss function.

MNIST

We train convolutional neural network with two convolutional layers (32 filters of size 5×5, 64 filters of size 5×5) and two fully-connected layers of 1024 and 10 units with ReLU activation. The output layer has 10 units with softmax activation.

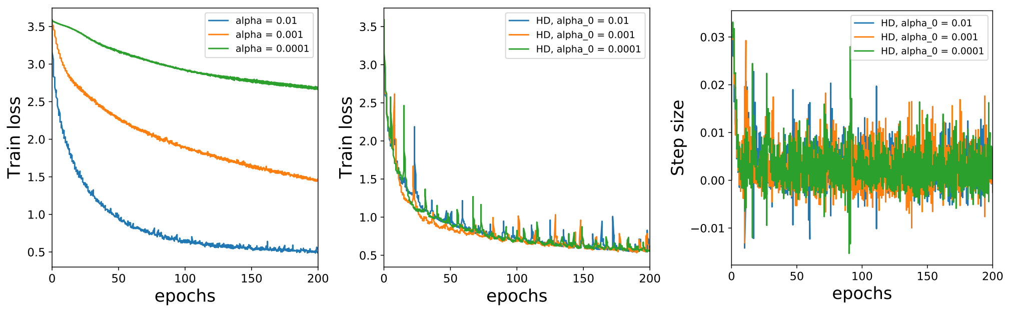
CIFAR-10 and SVHN

We train a convolutional neural network (CNN) with three convolutional layers (64 filters of size 5×5, 96 filters of size 3×3, and 128 filters of size 3×3) interspersed with max-pooling over 3×3 areas with stride 2. Two fully-connected layers with 512 and 256 units follow. We use ReLU activation function for all layers. The output layer has 10 units for the 10 classes of CIFAR-10 with softmax activation. The value of the $L2$-regularization parameter $\lambda = 0.002$.

6.2 Results

6.2.1 HD Adaptation for Momentum Optimization Algorithm

First, we want to demonstrate that HD adaptation can be applied to the Momentum optimization algorithm (without Nesterov acceleration) and gives a similar advantage in sensitivity to the initial step size selection Baydin et al., 2018. To show this we compare the Momentum with HD adaptation (1) and Momentum with constant step size (Figure 6.1). One can see that given the right β the learning of the model is the same for all initial step sizes α_0. Also, it can be seen that the optimal step size oscillates a lot and probably that causes short time increases of the train loss.



(A) Constant step size. (B) HD adaptation. (C) Step size.

Figure 6.1: Comparison between Momentum with constant step size and Momentum with HD adaptation on the CIFAR10 dataset (P3). For Momentum with HD adaptation, the regularization parameter $\beta = 0.001$ is the same as was used in Baydin et al., 2018. Given that β is near optimal for this problem, the optimization of the training loss is the same for all initial step sizes, whereas the performance of the Momentum with constant step size is quite sensitive to the step size.

6.2.2 Comparison Between PQ and HD Adaptation Algorithm with Fine-tuned β

We compare the performance of PQ and HD adaptation algorithms. First, we find the optimal regularization parameter β_{opt} for each adaptation model. We use grid search to find the optimal value of β_{opt}. In Figure 6.2 one can see the results for PQ and HD models on three different data sets. For all of them the training loss of the Moomentum with PQ adaptation is better that coresponding training loss of the Momentum with HD adaptation.

6.2.3 Sensitivity of the PQ and HD Adaptation Models to the Hyperparameters α_0 and β

For the second experiment, we compered the sensitivity of the Hypergradint Descent and PQ Model to the hyperparameters α_0 and β. For this we looked at the minimum value of the good estimate (e.g. computed on 1/5 of the whole training dataset) of the

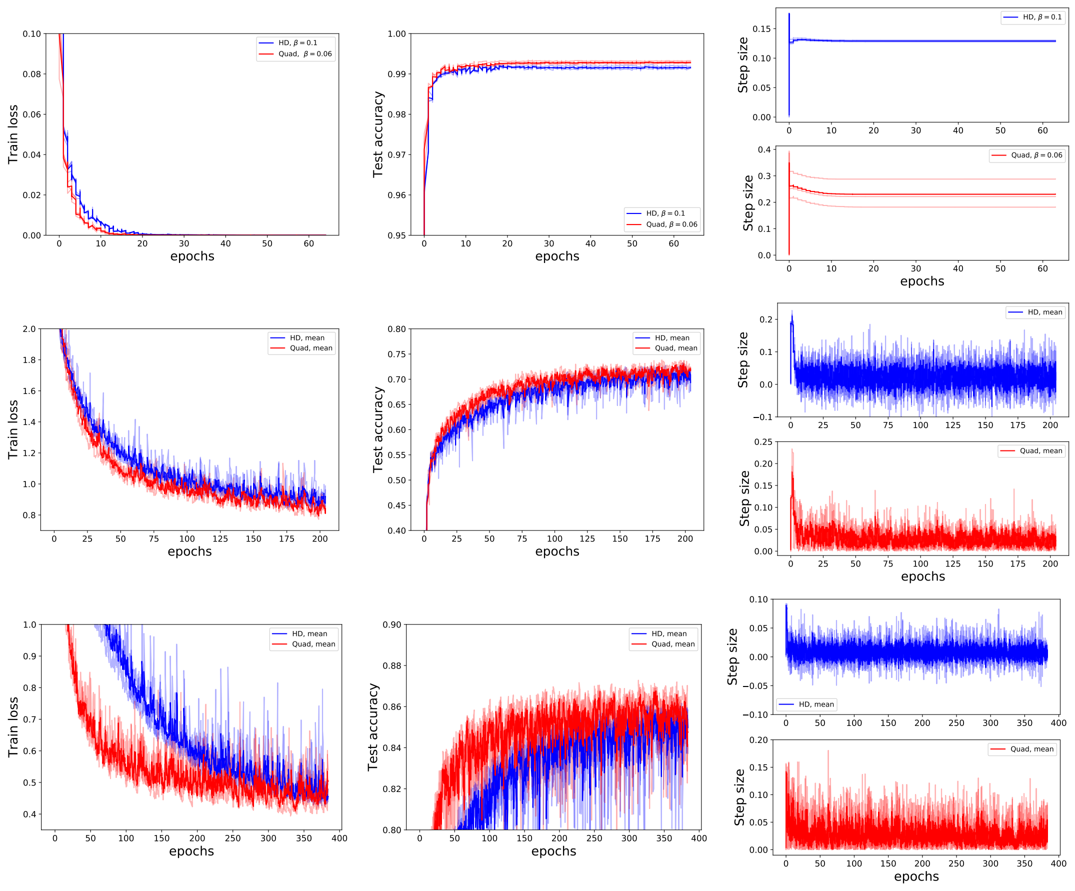


FIGURE 6.2: Experimental results of fine-tuned PQ (Algorithm 3) and HD (Algorithm 1) on three test problems. Rows: *top*: MNIST (P1); *middle*: SVHN (P2) *bottom*: CIFAR10 (P3). Columns: *right*: train loss; *middle*: test accuracy; *left*: step size. The parameter β was chosen by grid search. Initial step sizes are $\alpha_0 \in [10^{-2}, 10^{-3}, 10^{-4}]$ (transperent lines). Bold lines are the mean over all initial learning rates. PQ is always superior to the HD, dispite the fact that the same $\beta_{opt} = 0.06$ was used for the PQ model. For both Algorithm 1 and Algorithm 3 the value of momentum parameter is $\mu = 0.99$.

whole loss function. The minimum was chosen from the checkpoints computed during the optimization process of these algorithms on the grid of initial step sizes α_0 and hyperlearning rates β.

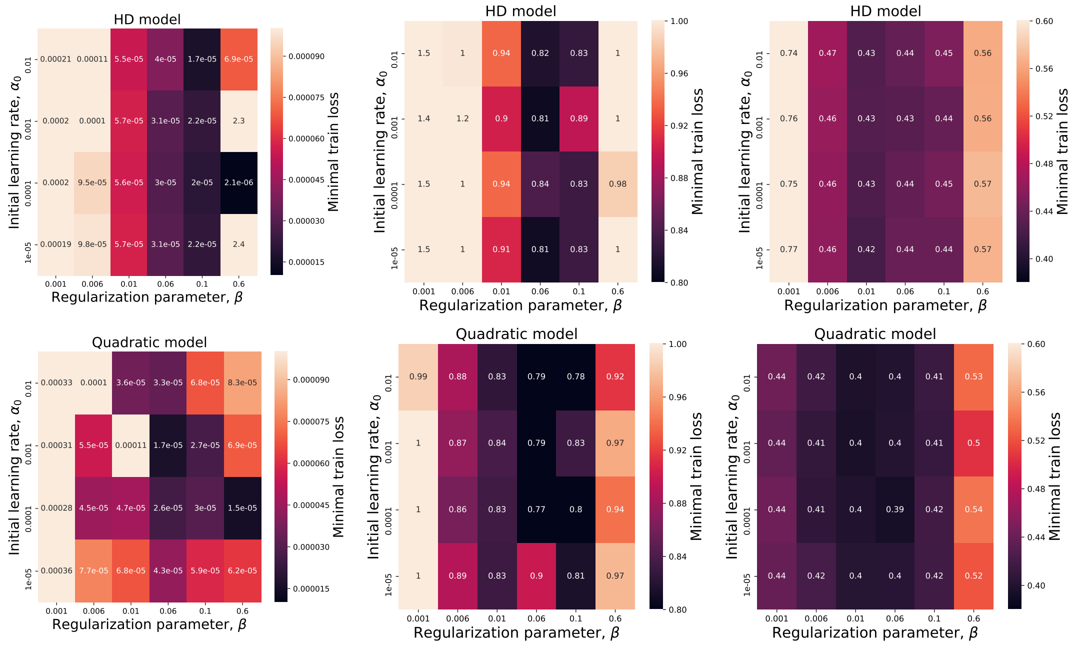


FIGURE 6.3: Comparison between PQ and HD adaptation. The minimum was chosen from the loss function values computed during the fixed number of optimization iterations that is enough for convergence. There are almost no sensitivity to the initial learning rate α_0. For the MNIST dataset, the difference is quite small as both algorithms are capable to find the minimum with quite small values. For SVHN and CIFAR10, the sensitivity to the β is smaller for PQ adaptation. The minimum loss is also smaller for PQ adaptation for any choice of hyperparameters α_0 and β. Rows: *top*: HD (Algorithm 1); *bottom*: PQ (Algorithm 3). Columns: *right*: MNIST (P1) ; *middle*: SVHN (P2); *left*: CIFAR10 (P3). For both Algorithm 1 and Algorithm 3 the value of momentum parameter is $\mu = 0.99$.

7 Conclusions

In this thesis, we studied the optimal choice of the step sizes schedule in the case of stochastic optimization. We introduced a proximal adaptation framework that uses a proximal operator of a convex model to adapt the step size. It was shown that hypergradient descent could be interpreted as proximal adaptation with a linear model. We developed PQ adaptation that aims to use all the available information to fit a quadratic model and then uses proximal point iteration for this model to adapt the step size. We compared two algorithms: Momentum with HD and PQ adaptation. For small datasets, both optimizers work well. On larger datasets, the PQ adaptation improves the performance of the optimization algorithm and is less sensitive to the hyperparameters choice.

Bibliography

Almeida, Luís B. et al. (1999). "Parameter Adaptation in Stochastic Optimization". *OnLine Learning in Neural Networks.* Cambridge University Press, pp. 111–134.

Balles, Lukas and Philipp Hennig (2018). "Dissecting ADAM: The Sign, Magnitude and Variance of Stochastic Gradients". *Proceedings of the 35th International Conference on Machine Learning.*

Barzilai, Jonathan and Jonathan M Borwein (1988). "Two-point step size gradient methods". *IMA journal of numerical analysis* 8.1, pp. 141–148.

Baydin, Atilim Gunes et al. (2018). "Online Learning Rate Adaptation with Hypergradient Descent". *International Conference on Learning Representations.*

Bengio, Yoshua (2012). "Practical recommendations for gradient-based training of deep architectures". *Neural networks: Tricks of the trade.* Springer, pp. 437–478.

Bergstra, James and Yoshua Bengio (2012). "Random search for hyper-parameter optimization". *Journal of Machine Learning Research*, pp. 281–305.

Bottou, Léon, Frank E. Curtis, and Jorge Nocedal (2018). "Optimization Methods for Large-Scale Machine Learning". *Siam Reviews* 60.2, pp. 223–311.

Boyd, Stephen and Lieven Vandenberghe (2004). *Convex optimization.* Cambridge university press.

Brockherde, Felix et al. (2017). "Bypassing the Kohn-Sham equations with machine learning". *Nature communications* 8.1, p. 872.

Chiu, Chung-Cheng et al. (2017). "State-of-the-art speech recognition with sequence-to-sequence models". *arXiv preprint arXiv:1712.01769.*

Defazio, Aaron, Francis Bach, and Simon Lacoste-Julien (2014). "SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives". *Advances in Neural Information Processing Systems*, pp. 1646–1654.

Delen, Dursun et al. (2013). "A comparative analysis of machine learning systems for measuring the impact of knowledge management practices". *Decision Support Systems* 54.2, pp. 1150–1160.

Goh, Gabriel (2017). "Why Momentum Really Works". *Distill.* URL: `http://distill.pub/2017/momentum`.

Golub, Gene H and James H Wilkinson (1966). "Note on the iterative refinement of least squares solution". *Numerische Mathematik* 9.2, pp. 139–148.

He, K. et al. (2016). "Deep Residual Learning for Image Recognition". *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778.

Johnson, Rie and Tong Zhang (2013). "Accelerating stochastic gradient descent using predictive variance reduction". *Advances in Neural Information Processing Systems*, pp. 315–323.

King, JR and S Dehaene (2014). "Characterizing the dynamics of mental representations: the temporal generalization method". *Trends in cognitive sciences* 18.4, pp. 203–210.

Kingma, Diederik P. and Jimmy Ba (2015). "ADAM: A Method for Stochastic Optimization". *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*. arXiv: `1412.6980`.

Koller, Daphne and Nir Friedman (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT press.

Kumar, Ankit et al. (2016). "Ask Me Anything: Dynamic Memory Networks for Natural Language Processing". *International Conference on Machine Learning*, pp. 1378–1387.

Lavecchia, Antonio (2015). "Machine learning approaches in drug discovery: methods and applications". *Drug discovery today* 20.3, pp. 318–331.

Maclaurin, Dougal, David Duvenaud, and Ryan Adams (2015). "Gradient-based hyperparameter optimization through reversible learning". *International Conference on Machine Learning*, pp. 2113–2122.

Mahsereci, Maren and Philipp Hennig (2015). "Probabilistic line searches for stochastic optimization". *Advances in Neural Information Processing Systems*, pp. 181–189.

Minka, Thomas P (2000). *Beyond newton's method.*

Parikh, Neal, Stephen Boyd, et al. (2014). "Proximal algorithms". *Foundations and Trends® in Optimization* 1.3, pp. 127–239.

Patrascu, Andrei and Ion Necoara (2017). "Nonasymptotic convergence of stochastic proximal point algorithms for constrained convex optimization". *arXiv preprint arXiv:1706.06297.*

Polyak, B.T. (1964). "Some methods of speeding up the convergence of iteration methods". *USSR Computational Mathematics and Mathematical Physics* 4.5, pp. 1 –17.

Schaul, Tom, Sixin Zhang, and Yann LeCun (2013). "No more pesky learning rates". *International Conference on Machine Learning*, pp. 343–351.

Schraudolph, N. N. (1999). "Local gain adaptation in stochastic gradient descent". *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470).* Vol. 2, 569–574 vol.2.

Snoek, Jasper, Hugo Larochelle, and Ryan P Adams (2012). "Practical Bayesian Optimization of machine learning algorithms". *Advances in Neural Information Processing Systems*, pp. 2951–2959.

Stuart, A. and J. K. Ord (1998). "Kendall's advanced theory of statistics. Vol.1: Distribution theory". *Kendall's advanced theory of statistics. Vol.1: Distribution theory, 6th ed. 6 Volumes. by A. Stuart and J.K. Ord. London: Hodder Arnold, 1998.* Halsted Press (Wiley, Inc.), p. 351.

Tan, Conghui et al. (2016). “Barzilai-Borwein Step Size for Stochastic Gradient Descent”. *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp. 685–693.

Tieleman, T. and G. Hinton. (2012). “RMSPROP. Divide the gradient by a running average of its recent magnitude.” *COURSERA: Neural networks for machine learning, Lecture 6.5.*

Vogt, Nina (2018). “Machine learning in neuroscience”. *Nature Methods* 15.1, p. 33.

Wolfe, Philip (1969). “Convergence conditions for ascent methods”. *SIAM review* 11.2, pp. 226–235.

Wu, Jian et al. (2017). “Bayesian optimization with gradients”. *Advances in Neural Information Processing Systems*, pp. 5267–5278.

A Maximum Likelihood Parameters

Using $\tilde{X}$ and $\tilde{y}$ defined in Equation (5.5) above we can compute maximum likelihood parameters for the model using Equation (5.8)

$$\hat{\theta} = a \cdot \begin{bmatrix} \left(-2\sigma_2^2\alpha f'(0) - 2\sigma_2^2\alpha f'(\alpha) + 4\sigma_2^2 f(\alpha) + f(0)\left(4\sigma_1^2 + \sigma_3^2\alpha^2 + \sigma_4^2\alpha^2\right)\right) \\ \left(-\sigma_4^2\alpha^2 f'(\alpha) - 2\sigma_4^2\alpha f(0) + 2\sigma_4^2\alpha f(\alpha) + f'(0)\left(4\sigma_1^2 + 4\sigma_2^2 + \sigma_3^2\alpha^2\right)\right) \\ \frac{1}{\alpha}\left(f'(0)\left(-2\sigma_1^2 - 2\sigma_2^2 - \sigma_3^2\alpha^2\right) + f'(\alpha)\left(2\sigma_1^2 + 2\sigma_2^2 + \sigma_4^2\alpha^2\right) + \right. \\ \left. + f(0)\left(-\sigma_3^2\alpha + \sigma_4^2\alpha\right) + f(\alpha)\left(\sigma_3^2\alpha - \sigma_4^2\alpha\right)\right) \end{bmatrix} \tag{A.1}$$

where $a = \frac{1}{4\sigma_1^2 + 4\sigma_2^2 + \sigma_3^2\alpha^2 + \sigma_4^2\alpha^2}$.
Under the Assumption 1, we can simplify Equation (A.1)

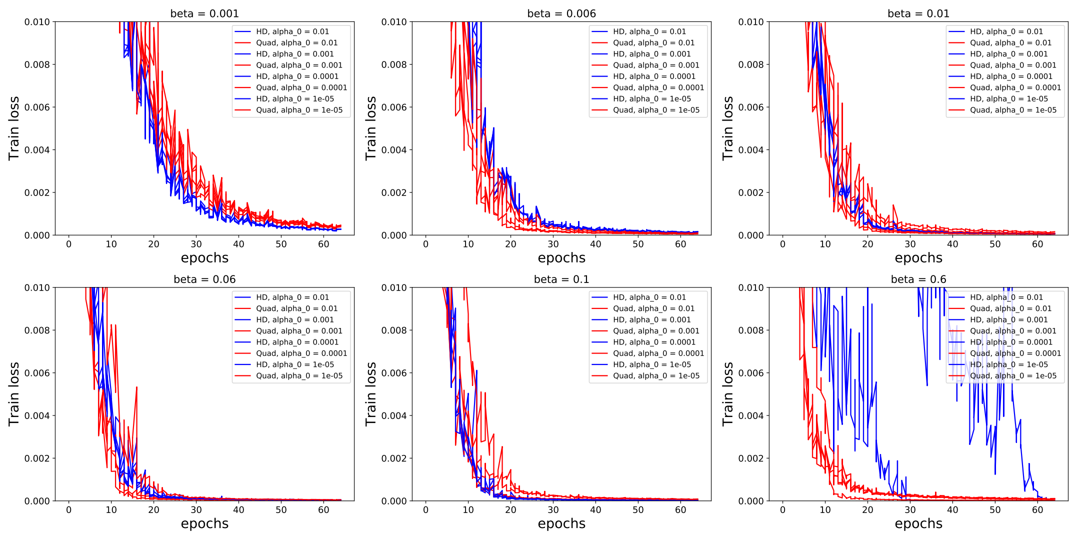
$$\hat{\theta} = \begin{bmatrix} \frac{1}{\sigma_{grad}^2\alpha^2 + 4\sigma_{loss}^2}\left(-\sigma_{loss}^2\alpha f'(0) - \sigma_{loss}^2\alpha f'(\alpha) + 2\sigma_{loss}^2 f(\alpha) + f(0)\left(\sigma_{grad}^2\alpha^2 + 2\sigma_{loss}^2\right)\right) \\ \frac{1}{\sigma_{grad}^2\alpha^2 + 4\sigma_{loss}^2}\left(-\frac{\sigma_{grad}^2 f'(\alpha)}{2}\alpha^2 - \sigma_{grad}^2\alpha f(0) + \sigma_{grad}^2\alpha f(\alpha) + \frac{f'(0)}{2}\left(\sigma_{grad}^2\alpha^2 + 8\sigma_{loss}^2\right)\right) \\ \frac{-f'(0) + f'(\alpha)}{2\alpha} \end{bmatrix}$$

If we assume that the variance is equal $\sigma_i = \sigma$ then we have

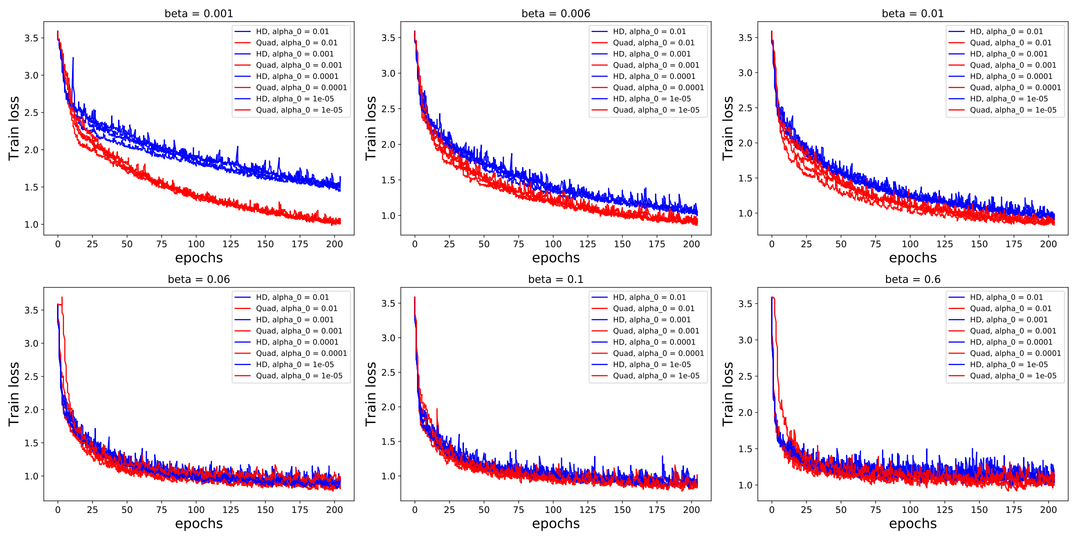
$$\hat{\theta} = \begin{bmatrix} \frac{1}{\alpha^2+4}\left(-\alpha f'(0) - \alpha f'(\alpha) + f(0)\left(\alpha^2 + 2\right) + 2f(\alpha)\right) \\ \frac{1}{\alpha^2+4}\left(-\frac{\alpha^2 f'(\alpha)}{2} - \alpha f(0) + \alpha f(\alpha) + \frac{f'(0)}{2}\left(\alpha^2 + 8\right)\right) \\ \frac{-f'(0) + f'(\alpha)}{2\alpha} \end{bmatrix}.$$

B Full Optimization Process

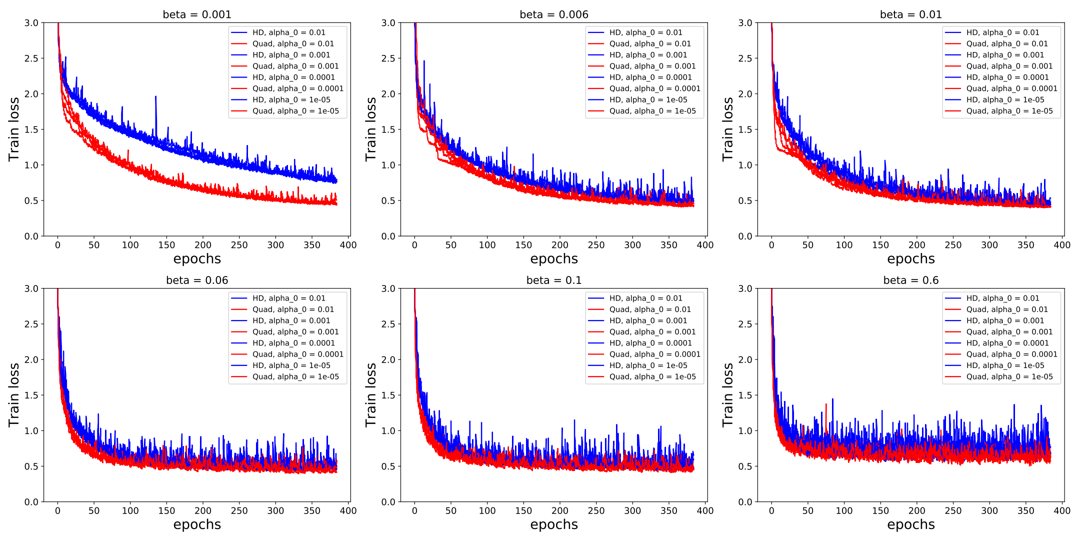
In the next page we presented the full optimization process for Momentum with HD and PQ adaptations. The sensitivity of the hyperaprameters were counted using the minimum values from these optimization process. We presented this plot to show that QP is better not only for minimum values for some fixed number of iterations, but during the whole optimization process.



(A) MNIST (P1)



(B) SVHN (P2)



(C) CIFAR10 (P3)

FIGURE B.1: Full optimization process that was used for estimation of sensitivity for Proximal Quadratic and Hypergradient Descent methods. Red lines are Proximal Quadratic and blue lines are Hypergradient Descent optimization process for different initial learning rates α_0.

C Quadratic Model Without Regularization

Here we present the pseudocode for a quadratic model without regularization. This adaptation is not stable because of a large bias in optimal step size estimate.

Algorithm 4 Momentum with Quadratic Adaptation

```
Require: initial parameter value $\theta_0$, initial step size $\alpha_0$, regularization constant $\beta$,
    momentum $\mu$, number of steps $T$
 1: Initialize $v = 0$, $m = 0$, $\alpha = \alpha_0$
 2: for $t = 1, \dots, T$ do
 3:     Evaluate stochastic gradient $g$
 4:     Evaluate one-dimentinal derivatives $\hat{f}'(\alpha) = g^T v$ and $\hat{f}'(0) = g_{old}^T v$
 5:     if  $0 \leq \frac{\hat{f}'(\alpha) - \hat{f}'(0)}{\alpha} \leq M$ or $f'(0) > 0$ then
 6:         Update $\alpha = -\frac{\alpha \hat{f}'(0)}{\hat{f}'(\alpha) - \hat{f}'(0)}$
 7:     end if
 8:     Update moving average $m = \mu m + (1 - \mu) g$
 9:     Evaluate new direction $v = -m$
10:     Update parameters $\theta = \theta + \alpha v$
11:     Update $g_{old} = g$
12: end for
```