ADAPTIVE MOMENTUM FOR NEURAL NETWORK OPTIMIZATION

ZANA RASHIDI

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

GRADUATE PROGRAM IN ELECTRICAL ENGINEERING & COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO

DECEMBER 2019

Abstract

In this thesis, we develop a novel and efficient algorithm for optimizing neural networks inspired by a recently proposed geodesic optimization algorithm. Our algorithm, which we call Stochastic Geodesic Optimization (SGeO), utilizes an adaptive coefficient on top of Polyak's Heavy Ball method effectively controlling the amount of weight put on the previous update to the parameters based on the change of direction in the optimization path. Experimental results on strongly convex functions with Lipschitz gradients and deep Autoencoder benchmarks show that SGeO reaches lower errors than established first-order methods and competes well with lower or similar errors to a recent second-order method called K-FAC (Kronecker-Factored Approximate Curvature). We also incorporate Nesterov style lookahead gradient into our algorithm (SGeO-N) and observe notable improvements. We believe that our research will open up new directions for high-dimensional neural network optimization where combining the efficiency of first-order methods and the effectiveness of second-order methods proves a promising avenue to explore.

Acknowledgements

I would like to sincerely thank my advisor, Dr. Aijun An, for her continuous support during my M.Sc. studies here at York University. I could not imagine a more passionate and caring advisor. I would also like to thank the members of my committee, Dr. Ruth Urner, and Dr. Michael Chen, for their encouragement and insightful comments, as well as Dr. Steven Wang, who has guided me through several projects, including my thesis, here at York. I thank all my friends, here, and all over the world, who made these two years much more enjoyable. Finally, I would like to thank my family, who have supported me unconditionally throughout my life.

Table of Contents

List of Figures

Chapter 1

Introduction

Neural networks are powerful function approximators which have seen much success in the past decade. In the realm of machine learning, neural networks are trained to learn nonlinear structures in multi-dimensional data such as images, audio, video and natural language. Even though neural networks have been around since the 1950s [1], where scientists were trying to model the brain, much of their success has been very recent due to the development of parallel processing units and abundance of data.

First order methods such as Stochastic Gradient Descent (SGD) with Momentum [2] and their variants are the methods of choice for optimizing neural networks. While there has been extensive work on developing second-order methods such as Hessian-Free optimization [3] and Natural Gradients [4, 5], they have not been successful in replacing first-order methods due to their large per-iteration costs of time and memory.

Although Nesterov's accelerated gradient and its modifications have been very effective in deep neural network optimization [2], some research have shown that Nesterov's method might perform suboptimally for strongly convex functions [6] without considering the local geometry of the function being optimized. Further, in order to get the best of both worlds, search for optimization methods which combine the efficiency of first-order methods and the effectiveness of second-order updates is still underway.

In this work, we introduce an adaptive coefficient for the momentum term on top of accelerated methods as an effort to combine first-order and second-order methods. We call our algorithms Geodesic Optimization (GeO) and Stochastic Geodesic Optimization (SGeO) (for the stochastic case) since it is inspired by a geodesic optimization algorithm proposed recently [7]. The adaptive coefficient effectively weights the momentum term based on the change in direction on the loss surface in the optimization process. The change in direction, which is calculate from a dot product, contributes as implicit local curvature information without resorting to expensive second-order information such as the Hessian or the Fisher Information Matrix.

Our experiments show the effectiveness of the adaptive coefficient on both strongly-convex functions with Lipschitz gradients and general non-convex problems, which are in our case, deep Autoencoders. GeO can speed up the convergence process significantly in convex

problems and SGeO can deal with ill-conditioned curvature such as local minima effectively as shown in our deep autoencoder benchmark experiments. SGeO has similar time-efficiency as first-order methods (e.g. Heavy Ball [8], Nesterov [9]) while reaching lower reconstruction error in fewer iterations. Compared to second-order methods (e.g., K-FAC [5]), SGeO has better or similar reconstruction error while being easier to implement and more memory-effective. We have also conducted generalization experiments and report the results.

The contributions of this thesis are summarized as follows:

- We introduce two novel algorithms based on an adaptive coefficient for first-order gradient momentum methods which speed up convergence for both convex and non-convex objective functions. The corresponding algorithms bridge the gap between simple but efficient first-order methods and effective but expensive second-order methods.
- We incorporate Nesterov's lookahead gradient method into the proposed algorithms successfully to further speedup convergence when optimizing an objective function.
- We demonstrate the effectiveness of the proposed methods via experiments on strongly convex functions and deep Autoencoder benchmarks while comparing to established baselines including a successful second-order method called K-FAC [5]. Our algorithms outperform first-order methods while having similar computational complexity, and show comparable performance with second-order methods, while being easier to implement and more memory efficient.
- We visualize the optimization path of the algorithm on global optimization benchmarks further validating our intuition.
- We analyze the behaviour of the adaptive coefficient in the course of optimization.

The structure of the thesis is as follows:

- **Chapter 2:** We give a background on the original geodesic and contour optimization introduced in [7], neural network optimization methods and the conjugate gradient method and discuss related work.
- **Chapter 3:** We introduce our adaptive coefficient specifically designed for convex problems and then modify it for non-convex cases.
- **Chapter 4:** Illustrates the algorithm's performance on convex and non-convex benchmarks as well as two global optimization benchmarks.
- **Chapter 5:** Concludes and summarizes our work and discusses future directions.

Chapter 2

Background and Related Work

We begin by reviewing the basics of neural networks and common practise in their optimization as well as new and related research in the area. Then we move on to conjugate gradient methods and discuss their extensions to non-quadratic problems. Finally, we give an overview of sequential geodesic optimization (SGEO) [7] and it's implications.

2.1 Neural Networks

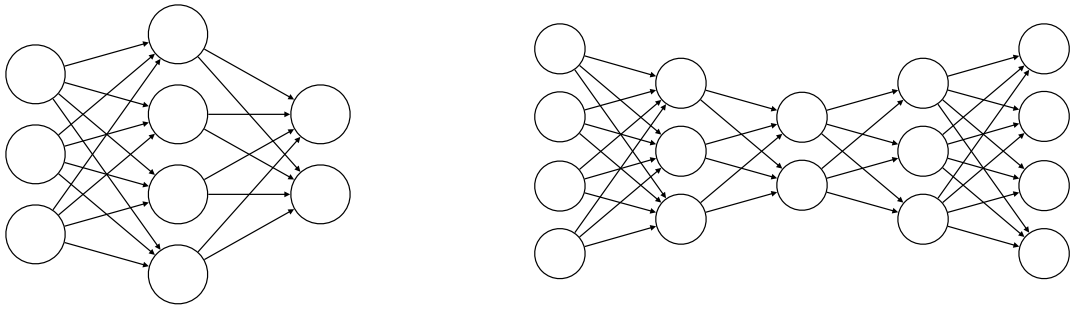
Neural networks are powerful models which have been very successful recently mostly because of the immense increase in the amounts of data and highly-efficient hardware [10]. A neural network is made up of a series of layers of varying sizes where each layer consists of several nodes. In a fully-connected feed-forward network, nodes in each layer are connected to all nodes in the next layer. Each node calculates a weighted average of the incoming values from the previous layers which is usually passed through a non-linearity such as the sigmoid function. An example of an arbitrary node's output after going through activation is the following:

$$a_{-,i} = \sigma(\sum_{k} w_{k,i} a_{k,i-1}) \tag{2.1}$$

where $w_{-,i}$ are the weights connected to that node, $a_{-,i-1}$ are the output values from the nodes connected to that node and σ is an activation function. Feed-forward neural networks can be used for classification and regression. A simple two layer neural network is shown in Figure 2.1 where the lines connecting the nodes are the weights ($w_{-,i}$).

Other task-specific architectures for neural networks also exist such as convolutional neural networks (CNNs) where convolutional layers are used to scan multi-dimensional data such as images or videos. Recurrent neural networks (RNNs) on the other hand, use many layers of recurrent nodes which are suited for sequential data such as natural language. Autoencoders are neural networks that are traditionally used for dimensionality reduction. The autoencoder consists of two parts, first, the encoder, which encodes the input through a series of layers ending with a bottleneck layer and the decoder, which tries to reconstruct the input from the

representation learned in the bottleneck layer. See Figure 2.1 for an example. Autoencoders have been recently used in generative models as well [11].



(a) Feed-forward Neural Network. (b) Autoencoder.

Figure 2.1: Sample Feed-forward (a) and Autoencoder (b) neural network architectures.

2.1.1 Optimization

Regardless of the task given to a neural network, the goal is to minimize an empirical loss function which estimates how far off we are from the distribution of the existing data. By changing the parameters (weights) of the neural network we aim to find parameters such that the loss function is minimized. We consider a neural network with a differentiable loss function $f : \mathbb{R}^D \to \mathbb{R}$ with the set of parameters θ where θ is the collection of all the weights in the neural network. The objective is to minimize the loss function f with a set of iterative updates to the parameters θ. The respective loss function can take many forms, commonly the cross-entropy loss or the squared error loss.

There has been extensive work on large-scale optimization techniques for neural networks in recent years. A good overview can be found in [12]. Here we discuss established first-order methods and some related second-order methods.

2.1.2 First-order Methods

The most common way of constructing the update to optimize the loss function $f(\theta)$ is to move in the opposite direction of the gradient (giving us the steepest descent) or a linear combination of existing gradient information from past updates. These methods are called first-order methods since we're only using first-order information from the function.

The simplest approach is gradient descent which proposes the following update:

$$\theta_{t+1} = \theta_t - \epsilon \nabla f(\theta_t) \tag{2.2}$$

where θ_t and θ_{t+1} are the weights from two consecutive iterations, $\nabla f(\theta_t)$ is the gradient of the function at the point θ and ϵ is the learning rate. However, this update can be very slow and choosing an appropriate learning rate ϵ can be hard. A large learning rate can cause oscillations and overshooting. On the other hand, a small learning rate can slow down the convergence drastically.

2.1.3 Acceleration

Several recent and previous works have focused on acceleration for first-order methods. Accelerated methods utilize the information stored from previous updates. The intuition being that a running average of the past gradients is a better guide than the local gradient.

2.1.4 Heavy Ball method

Polyak suggested that in order to speed up the convergence of gradient descent, one can add a momentum term where we store an average of past updates [8].

$$\begin{aligned} d_{t+1} &= \mu d_t - \epsilon \nabla f(\theta_t) \\ \theta_{t+1} &= \theta_t + d_{t+1} \end{aligned} \tag{2.3}$$

where d is the previous update (called the velocity) and μ is the coefficient (called the momentum parameter). Polyak's physical analogy for the algorithm was a heavy ball rolling down a hill gaining momentum over time, thus the name of the algorithm.

2.1.5 Nestrov's method

Nesterov's accelerated gradient [9] can be rewritten as a momentum method ([2]):

$$\begin{aligned} d_{t+1} &= \mu d_t - \epsilon \nabla f(\theta_t + \mu d_t) \\ \theta_{t+1} &= \theta_t + d_{t+1} \end{aligned} \tag{2.4}$$

Nesterov's momentum is different from the heavy ball method only in where we take the gradient, resulting in a "lookahead" gradient. For the class of smooth and convex functions, it has a convergence rate of $O(1/T^2)$ [9] versus the $O(1/T)$ convergence rate for gradient descent. Note that in both of these methods d_t is the previous update $\theta_t - \theta_{t-1}$.

More recently, the authors in [13] propose an adaptive method to accelerate Nesterov's algorithm in order to close a small gap in its convergence rate for strongly convex functions with Lipschitz gradients adding a possibility of more than one gradient call per iteration.

In [14], the authors propose a differential equation for modeling Nesterov's algorithm inspired by the continuous version of gradient descent, a.k.a. gradient flow. [15] take this further and suggest that all accelerated methods have a continuous time equivalent defined by a Lagrangian functional, which they call the Bregman Lagrangian. They also show that acceleration in continuous time (as opposed to discrete time-steps in optimization) corresponds to traveling on the same curve in spacetime at different speeds.

In a recent work, [16] proposes a differential geometric interpretation of Nesterov's method for strongly-convex functions with links to continuous time differential equations mentioned earlier and their Euler discretization.

2.1.6 Gradient Descent Variants

Adagrad [17] is an optimization technique that extends gradient descent and adapts the learning rate according to the neural network parameters. Adadelta [18] and RMSprop [19] improve upon Adagrad by reducing its aggressive deduction of the learning rate. Our method is differnet from these methods since we use momentum in our algorithm and we set the learning rate according to a schedule instead of an adaptive learning rate. Our adaptive coefficient is used for the momentum term and not the learning rate. Adam [20] improves upon the previous methods by keeping an additional average of the past gradients which is similar to what regular momentum does. Our method does not keep a running average of past squared gradients and employs an adaptive coefficient for the momentum term while Adam uses a constant value instead. Adaptive Restart [21] proposes to reset the momentum whenever rippling behaviour is observed in accelerated gradient schemes. AggMo [22] keeps several velocity vectors with distinct parameters in order to damp oscillations. AMSGrad [23] on the other hand, keeps a longer memory of the past gradients to overcome the suboptimality of the previous algorithms on simple convex problems. Our method only keeps one velocity vector, however, we keep track of the optimization path via the adaptive coefficient.

2.1.7 Second-order Methods

Second-order methods are usually variants of Newton-Rhapson (or simply Newton)'s method where we have the following update based on a second-order local Taylor approximation of the function:

$$\theta_{t+1} = \theta_t - H(\theta_t)^{-1} \nabla f(\theta_t) \tag{2.5}$$

where H^{-1} is the inverse of the Hessian matrix. However, calculating and storing the Hessian can be prohibitive in high-dimensional neural network optimization. Conversely, second-order methods are desirable because of their fine convergence properties due to dealing with bad-conditioned curvature by using local second-order information.

Hessian-Free optimization [3] is based on the truncated-Newton approach where the conjugate gradient algorithm is used to optimize the quadratic approximation of the objective function. The natural gradient method [4] reformulates the gradient descent in the space of the prediction functions instead of the parameters. That is, instead of minimizing the negative likelihood as in gradient descent, we formulate the problem as a probabilistic model and maximize the likelihood, $p(x|\theta)$. Since the likelihood is a probability distribution, we take the steepest descent step in the space of distributions instead of parameters. The metric used to measure the distance between two probability distributions is KL-divergence and the Hessian of the KL-divergence between two distributions is the Fisher information matrix.

K-FAC K-FAC [5] approximates the Fisher information matrix which appears in the natural gradient method. The method works by reducing blocks of the Fisher information matrix to kronecker products of smaller matrices. Even though K-FAC's approximation are still expensive to run, but it has proven effective in optimizing neural networks.

Our method is different from second-order methods since we are not using explicit second-order information but rather implicitly deriving curvature information using the change in direction.

2.2 Conjugate Gradients

Conjugate gradient methods can be considered as middle-ground to steepest descent methods (first-order) and Newton's method (second-order). As discussed before, steepest descent provides a very slow update while Newton's method provide effective updates but they need to calculate the inverse of the Hessian. Conjugate gradients were first introduced as way to solve linear programming problems [24] (equivalent to quadratic problems with positive definite matrices) but were later extended to non-linear cases [25].

2.2.1 Conjugate Directions Theorem

Consider the following quadratic problem:

$$\min_{\theta \in \mathbb{R}^n} \frac{1}{2}\theta^T Q\theta - b^T\theta \tag{2.6}$$

where Q is a $n \times n$ positive definite matrix. The unique solution to the above problem is also the unique solution to the following linear programming problem:

$$Q\theta = b \tag{2.7}$$

where $\theta \in \mathbb{R}^n$. We denote the unique solution by θ^* where:

$$\theta^* = \epsilon_1 d'_1 + ... + \epsilon_n d'_n \tag{2.8}$$

where $\{d'_1, ..., d'_n\}$ are conjugate vectors w.r.t the matrix Q, i.e.:

$$\forall i, j \in \{1, ..., n\} : \quad d'_i Q d'_j = 0 \tag{2.9}$$

indicating that the vectors can be considered as basis vectors for $\mathbb{R}^n$.

The conjugate directions theorem states that if $\theta_1 \in \mathbb{R}^n$ is an arbitrary vector, using the following rules for n steps:

$$\begin{aligned} \theta_{t+1} &= \theta_t + \epsilon_t d'_t \\ g_t &= Q\theta_t - b \\ \epsilon_t &= -\frac{g_t^T d'_t}{{d'_t}^T Q d'_t} \end{aligned} \tag{2.10}$$

we'll have $\theta_{n+1} = \theta^*$. This means that for quadratic problem, assuming we have a way to calculate the conjugate vectors ($\{d'_1, ..., d'_n\}$), the former algorithm will converge in n steps where n is the dimension.

2.2.2 The Conjugate Gradient Algorithm

The conjugate gradient algorithm proposes a way to calculate the Q-conjugate vectors $\{d'_1, ..., d'_n\}$. Based on [24] the conjugate gradient update is:

$$\begin{aligned} \theta_{t+1} &= \theta_t + \epsilon_t d'_t \\ d'_{t+1} &= -g_{t+1} + \gamma_t d'_t \end{aligned} \tag{2.11}$$

where ϵ and γ are step sizes and d' is the search direction. Using the conjugate direction theorem, one can easily prove that the conjugate gradient algorithm is a conjugate direction method. That is, by following the conjugate gradient update, we reach the solution in n steps.

2.2.3 Non-quadratic Functions

For a non-quadratic function f where we want to solve $\min_{\theta \in \mathbb{R}^n} f(\theta)$, one can approximate the function locally using a quadratic where:

$$\begin{aligned} g_t &= \nabla f(\theta_t) \\ Q &= \nabla^2 f(\theta_t) \end{aligned} \tag{2.12}$$

and ϵ and γ are calculate in the following way:

$$\begin{aligned} \epsilon_t &= -\frac{g_t^T d'_t}{d_t'^T [\nabla^2 f(\theta_t)] d'_t} \\ \gamma_t &= \frac{g_{t+1}^T [\nabla^2 f(\theta_t)] d'_t}{d_t'^T [\nabla^2 f(\theta_t)] d'_t} \end{aligned} \tag{2.13}$$

The non-quadratic case of conjugate gradients is very similar to Newton's method where the function is locally approximated by a quadratic function. As mentioned in the previous section, Newton's method approximates the function locally with a quadratic function. In the standard version of the Newton's method, this approximation is equivalent to a second-order Taylor approximation. Further, although line search is not required, similar to Newton's method, the Hessian still needs to be calculated at each point.

Note that since at each point the non-linear CG algorithm optimizes a quadratic, the algorithm has to restart after n steps and continue from the last point of the previous optimization until a condition is satisfied.

2.2.4 Approximations

To avoid calculating the Hessian $\nabla^2 f$ which can be very expensive in terms of computation and memory, ϵ is usually determined using a line search, i.e. by approximately calculating

$$\epsilon_t = \arg\min_{\epsilon} f(\theta_t + \epsilon d_t) \tag{2.14}$$

and several approximations to γ have been proposed. For example, [25] (Fletcher-Reeves) have proposed the following:

$$\gamma_t^{FR} = \frac{\|g_t\|^2}{\|g_{t-1}\|^2} \tag{2.15}$$

Note that γ^{FR}, if used with an exact line search, is equivalent to the original conjugate gradient algorithm in the quadratic case.

2.2.5 Conjugate Gradients as Momentum

Rearranging the update, one can see the conjugate gradient method as a momentum method where ϵ_t is the learning rate and $\epsilon_t \gamma_{t-1}$ is the momentum parameter:

$$\theta_{t+1} = \theta_t - \epsilon_t g_t + \epsilon_t \gamma_{t-1} d'_{t-1} \tag{2.16}$$

An interesting observation here is that the momentum parameter that appears in the equation above, is adaptively changing. This change can vary based on the particular approximation used. As an example, the Fletcher-Reeves approximation [25] measures the change in the magnitude of the gradient in consecutive updates.

Note that $d'_t = (\theta_{t+1} - \theta_t)/\epsilon_t$. We added the prime notation to avoid confusion with $d_t = \theta_{t+1} - \theta_t$ throughout the thesis.

2.3 Geodesic and Contour Optimization

The goal is to solve the optimization problem $\min_{\theta \in \mathbb{R}} f(\theta)$ where $f : \mathbb{R}^D \to \mathbb{R}$ is a differentiable function. The authors in [7] approach the problem by following the geodesics (roughly, shortest paths on a curve) on the loss surface guided by the gradient. In order to derive the geodesic equation, one needs to study differential geometry which is out of the scope of this thesis, thus we only give a brief introduction here. There are numerous comprehensive resources on this topic [26].

2.3.1 Differential Geometry

Differential geometry is the building block of the theory of General Relativity where spacetime is a 4-dimensional Lorentzian manifold and geodesics describe the path of free falling particles under the influence of massive objects [26].

2.3.2 Geodesics Equation

The geodesic equation using local coordinates $X^i(t)$ and the Einstein summation convention can be characterized as:

$$\frac{d^2X^i(t)}{dt^2} + \Gamma^i_{jk}\frac{d^2X^j(t)}{dt}\frac{d^2X^k(t)}{dt} = 0 \tag{2.17}$$

where Γ^i_{jk} are the Christoffel symbols. Christoffel symbols are the connections coefficients of the so called Levi-Civita connection ${}^{L.C.}\nabla$ and are defined as:

$$\Gamma^i_{jk} = \frac{1}{2}g^{im}\Big(\frac{\partial g_{mj}}{\partial x^k} + \frac{\partial g_{mk}}{\partial x^j} - \frac{\partial g_{jk}}{\partial x^m}\Big) \tag{2.18}$$

where g_{ij} is the metric and g^{ij} is the inverse metric. The Levi-Civita connection is a torsion-free metric-preserving connection that is unique according to the fundamental theory of Riemannian Geometry.

2.3.3 Quadratic Approximation

Calculating the geodesics however, requires the inversion of the metric which appears in the christoffel symbols. They overcome this issue using conformal mapping where they perform the calculations in $\mathbb{R}^D$ (where the metric is the Kronecker delta) and map the results to the original manifold. A conformal map is a map that preserves angles and orientations locally. The following theorem about the geodesics under conformal mapping was proven in [7]:

Theorem (3.1) *The geodesics on any manifold conformally related to the Euclidean space, are asymptotically parallel to either the gradient or the level curves of the objective function f* [7].

Further, in order to solve the geodesic equation iteratively, the authors approximate it using a quadratic function [7] under conformal mapping. In the neighbourhood of θ_t, the solution of the geodesic equation can be approximated as:

$$\theta_{t+1} = \theta_t + v_t\delta_t - c_t\delta_t^2 \tag{2.19}$$

where θ_t is the current point, v_t is the unit tangent vector of the geodesic at θ_t, δ_t is the step size and:

$$c_t = \frac{1}{2}\big[\nabla f(\theta_t) - 2(v_t \cdot \nabla f(\theta_t))v_t\big] \tag{2.20}$$

and θ_{t+1} is the next point. The tangent vector is defined as the normalized difference vector between the current point and the previous point

$$v_t = \frac{\theta_t - \theta_{t-1}}{\|\theta_t - \theta_{t-1}\|} \tag{2.21}$$

and the first tangent vector is set to the gradient $v_1 = \frac{\nabla f(\theta_1)}{\|\nabla f(\theta_1)\|}$.

For a detailed explanation we refer the reader to the original publication [7].

2.3.4 Rewriting the Geodesic Update

Similar to what we did for the conjugate gradient method, denoting $\nabla f(\theta_t)$ as g_t, the geodesic update can be rewritten as a momentum method:

$$\begin{aligned} v_{t+1} &= \delta_t \left[\Big(1 + \delta_t \big(v_t \cdot g_t \big) \Big) v_t - \frac{\delta_t}{2} g_t \right] \\ \theta_{t+1} &= \theta_t + v_{t+1} \end{aligned} \tag{2.22}$$

where v_t is calculated according to Equation 2.21. Comparing this with the Heavy Ball method (2.3), we see an adaptive coefficient $1 + \delta_t \big(v_t \cdot g_t \big)$ for the velocity term which we will analyse in detail in the next chapter. Since both the update and the gradient are normalized, the adaptive coefficient reduces to a cosine term:

$$1 + \delta_t \big(v_t \cdot g_t \big) = 1 + \delta_t \cos \left(\pi - \phi_t \right) \tag{2.23}$$

where ϕ_t is the angle between v_t and $-g_t$.

It is worth noting that non-linear conjugate gradient methods (2.2) employ similar coefficients for the step sizes of the search direction. Both methods try to embed non-expensive implicit second order information, based on changes in direction (e.g. gradients).

Further, SGEO is introduced as a global optimization algorithm that employs several line searches, local optimization (e.g. Quasi-Newton) and random walks that transfer from one local optima to another [7]. We shall address these issues in detail in the next chapter.

2.4 Convex Functions

Functions can be grouped based on various factors. Generally, convexity is an important factor that divides optimization into convex optimization and non-convex optimization, where the latter is a hard problem. Here we introduce some simple concepts that will be needed in the following chapters

Convexity A function $f : \mathbb{R}^D \to \mathbb{R}$ is convex when its domain is a convex set and for all θ_1 and θ_2 in the domain and all $\lambda \in [0, 1]$ we have:

$$f\big(\lambda \theta_1 + (1 - \lambda) \theta_2 \big) \leq \lambda f(\theta_1) + (1 - \lambda) f(\theta_2) \tag{2.24}$$

Strong-Convexity Since we will be experimenting with strongly-convex functions with Lipschitz gradients, we introduce the basics of such a function. A $\ddot{\mu}$-strongly convex function $f : \mathbb{R}^D \to \mathbb{R}$ has the following property:

$$f(\theta_2) \geq f(\theta_1) + f'(\theta_1) \cdot (\theta_2 - \theta_1) + \frac{\ddot{\mu}}{2} \|\theta_2 - \theta_1\|_2^2 \tag{2.25}$$

Lipschitz-ness A function f has L-Lipschitz gradients when:

$$\|f'(\theta_1) - f'(\theta_2)\|_2 \leq L\|\theta_1 - \theta_2\|_2^2 \tag{2.26}$$

Chapter 3

Stochastic Geodesic Optimization

3.1 Adaptive Coefficient for Momentum

In this chapter we introduce an adaptive coefficient for Polyak's heavy ball algorithm based on Sequential Geodesic Optimization (SGEO) algorithm introduced in the previous chapter. Here we review and discuss the components used in constructing the algorithms and then introduce the algorithms.

3.1.1 Sequential Geodesic Optimization

As discussed in the previous chapter, SGEO introduces a way to follow the geodesics (guided by the gradient) on the manifold of a function via Equation 2.22:

$$
\begin{aligned}
v_{t+1} &= \delta_t \left[\Big(1 + \delta_t \big(v_t \cdot g_t \big) \Big) v_t - \frac{\delta_t}{2} g_t \right] \\
\theta_{t+1} &= \theta_t + v_{t+1}
\end{aligned}
$$

where a dot product between the previous update and the gradient $1 + \delta_t(v_t \cdot g_t)$ appears as a step size for the existing search direction.

3.1.2 Issues with SGEO

There are many challenges is adapting SGEO for optimizing high-dimensional (loss) functions such as that of neural networks. Here we point out some of the challenges:

Full-gradient information The algorithm utilizes full-gradient information. However, extracting full-gradient information is not practical when optimizing neural networks since large datasets are used to calculate the gradient. A solution is to use partial gradient information by sampling random mini-batches which is more efficient.

Local searches In the original publication [7], SGEO is made up of two phases, first, the algorithm finds a basin of attraction using geodesics, then, it uses an arbitrary local search (Quasi-Newton) to search for optima in that region. This is also not practical when applied to neural networks as local searches are usually expensive procedures. Further, when combined with partial gradient information, it is very likely that a local optimization process will overfit the current mini-batch. Thus we reduce the optimization process to one stage.

Adaptive step size The authors in [7] propose an adaptive step size based on the ratio of the quadratic term and the linear term of the approximation:

$$\delta_t = \frac{1}{N} \sum_{i=1}^{N} \left| \frac{v_{ti}}{c_{ti}} \right| \tag{3.1}$$

where N is the dimension. However, again to due partial gradient information, this step size makes training neural networks very unstable. We use a simple multiplicative step size as an alternative discussed in section 4.3.2.

3.1.3 Adaptive Coefficient Intuition

The dot product in the adaptive coefficient that appears before the unit tangent vector in equation 2.22 has an intuitive geometric interpretation:

$$\bar{g}_t \cdot \bar{d}_t = \cos(\pi - \phi_t) \quad \text{where} \quad \bar{g}_t = \frac{g_t}{\|g_t\|}; \quad \bar{d}_t = \frac{d_t}{\|d_t\|} \tag{3.2}$$

where ϕ_t is the angle between the previous update $d_t = \theta_t - \theta_{t-1}$ and the negative of the current gradient $-g_t$. Since $0 \leq \phi \leq \pi$, thus

$$-1 \leq \bar{g}_t \cdot \bar{d}_t = \cos(\pi - \phi_t) \leq 1 \quad \text{(Figure 3.1 (b))} \tag{3.3}$$

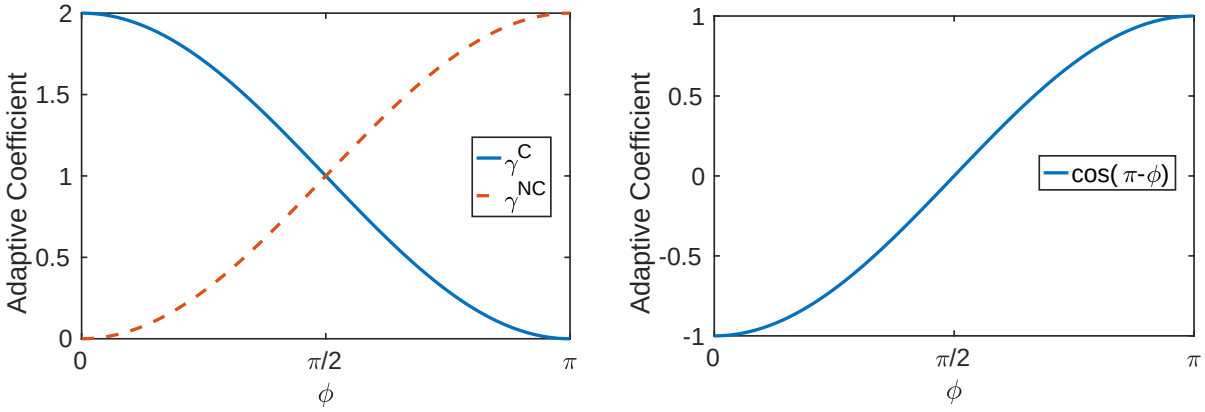
The adaptive coefficient embeds a notion of change of direction of the optimization path. This notion can be interpreted as implicit second-order information where it tells us how much the current gradient's direction is different from the previous gradients which is similar to what second-order information (e.g. the Hessian) provide:

$$H(\theta_t) = \nabla^2 f(\theta_t) \tag{3.4}$$

The Hessian provides the rate of change in the gradient of a function at a point while the gradient tells us the rate of change of the function itself. Since the previous update contains a running estimate of the past gradients, the dot product is basically comparing the current gradient against all aggregated past gradients.

3.2 Convex Functions

Here, we design a new algorithm based on the proposed adaptive coefficient derived from geodesics for convex functions.



(a) Adaptive coefficient

(b) Cosine value

Figure 3.1: Adaptive Coefficient and cosine (dot product) values for the convex $\gamma^C = 1 - \bar{g} \cdot \bar{d}$ and non-convex $\gamma^{NC} = 1 + \bar{g} \cdot \bar{d}$ cases where $\bar{g} \cdot \bar{d}$ is equal to $\cos(\pi - \phi)$.

3.2.1 Formulation and Intuition

We propose to apply this implicit second-order information to the Heavy Ball method of [8] as an adaptive coefficient for the momentum term. The formulation of a coefficient for a convex function should be such that: we reinforce the effect of the previous update when the directions align, i.e. in the extreme case: $\phi = 0$ and decrease when they don't, i.e. the other extreme case: $\phi = \pi$. In other words, the algorithm takes bigger steps when moving down a steep hill and slows down when the manifold of the function starts to curve.

Thus, we write the coefficient as

$$\gamma_t^C = 1 - \bar{g}_t \cdot \bar{d}_t \quad \text{(Figure 3.1 (a))} \tag{3.5}$$

with C indicating "convex" which is used in line 6 of Algorithm 1. It's obvious that $0 \leq \gamma_t^C \leq 2$ since $-1 \leq \bar{g}_t \cdot \bar{d}_t \leq 1$. When $\phi = 0$, i.e. when directions align, $\gamma^C = 2$, which results in bolder step. Similarly when $\phi = \pi/2$, i.e. when the vectors are orthogonal, $\gamma^C = 1$ and when $\phi = \pi$, when directions are opposite, $\gamma^C = 0$. The last case might happen when we have stepped over an optima, telling us to trust the gradient more which results in stepping back to the optima. Note that we will use the bar notation (e.g. $\bar{d}$) throughout the thesis indicating normalization by magnitude.

Applying the proposed coefficient to the Heavy Ball method, we have the following algorithm which we call GeO (Geodesic Optimization):

Algorithm 1: GeO (convex)

1 Initialize θ_1
2 Set $d_1 = \nabla f(\theta_1)$
3 **for** $t = 1$ ***to*** T **do**
4 Calculate the gradient $g_t = \nabla f(\theta_t)$
5 Calculate adaptive coefficient $\gamma_t^C = 1 - \bar{g}_t \cdot \bar{d}_t$
6 Calculate update $d_{t+1} = \alpha\gamma_t^C d_t - \epsilon g_t$
7 Update parameters $\theta_{t+1} = \theta_t + d_{t+1}$

where T is total number of iterations, $\alpha \in [0, 1]$ is a tunable parameter set based on the function being optimized and ϵ is the learning rate. The algorithm is different from the Heavy Ball method only in line 6 where we add our geodesic coefficient.

3.2.2 Incorporating Nesterov's momentum

We can further incorporate Nesterov's lookahead gradient into GeO by modifying line 4 to

$$g_t = \nabla f(\theta_t + \mu d_t) \tag{3.6}$$

which is to take the gradient at a further point $\theta_t + \mu d_t$ where μ is a tunable parameter usually set to a value close to 1. We call this algorithm GeO-N:

Algorithm 2: GeO-N (convex)

1 Initialize θ_1
2 Set $d_1 = \nabla f(\theta_1)$
3 **for** $t = 1$ ***to*** T **do**
4 Calculate the gradient $g_t = \nabla f(\theta_t + \mu d_t)$
5 Calculate adaptive coefficient $\gamma_t^C = 1 - \bar{g}_t \cdot \bar{d}_t$
6 Calculate update $d_{t+1} = \alpha\gamma_t^C d_t - \epsilon g_t$
7 Update parameters $\theta_{t+1} = \theta_t + d_{t+1}$

3.3 Non-convex functions

In this section, we tweak our adaptive coefficient to adapt to non-convex functions since our focus is neural network optimization which involves highly non-convex loss functions.

3.3.1 Formulation and Intuition

The algorithm proposed in the previous section would be problematic for non-convex functions such as the loss function when optimizing neural networks. Even if the gradient information was not partial (due to minibatching), the current direction of the gradient cannot be trusted

because of non-convexity and poor curvature (such as local minima, saddle points, etc). To overcome this issue, we propose to alter the adaptive coefficient to

$$\gamma_t^{NC} = 1 + \bar{g}_t \cdot \bar{d}_t \quad \text{(Figure 3.1 (a))} \tag{3.7}$$

with NC indicating "non-convex".

By applying this change, we are reinforcing the previous direction when the directions do not agree thus avoiding sudden and unexpected changes of direction (i.e. gradient). In other words, we choose to trust the previous history of the path already taken more, thus acting more conservatively.

Considering the extreme cases, for instance, when $\phi = 0$, when directions align, $\gamma^{NC} = 0$, resulting in trusting the gradient while acting more cautious than the convex case, by taking a smaller step. Similarly when $\phi = \pi/2$, the orthogonal case, $\gamma^{NC} = 1$, which is similar to the convex case. However, when $\phi = \pi$, opposite directions, $\gamma^{NC} = 2$, which might happen when the algorithm hits a local optima. In this case, we trust the previous update more, resulting in better recovery from local minima/maxima.

To increase efficiency, we use minibatches, calling the following algorithm SGeO (Stochastic Geodesic Optimization):

Algorithm 3: SGeO (NON-CONVEX)

1 Initialize θ_1
2 Set $d_1 = \nabla f(\theta_1)$
3 **for** $t = 1$ ***to*** T **do**
4 Draw minibatch from training set
5 Calculate the gradient $g_t = \nabla f(\theta_t)$
6 Calculate adaptive coefficient $\gamma_t^{NC} = 1 + \bar{g}_t \cdot \bar{d}_t$
7 Calculate update $d_{t+1} = \epsilon_t \gamma_t^{NC} \bar{d}_t - \epsilon_t \bar{g}_t$
8 Update parameters $\theta_{t+1} = \theta_t + d_{t+1}$

Further, we found that using the unit vectors for the gradient $\bar{g}$ and the previous update $\bar{d}$, when calculating the next update in the non-convex case makes the algorithm more stable. In other words, the algorithm behaves more robustly when we ignore the magnitude of the gradient and the momentum term and only pay attention to their directions (line 7). Thus, the magnitudes of the updates are solely determined by the corresponding step sizes, which are in our case, the learning rate and the adaptive geodesic coefficient.

3.3.2 Incorporating Nesterov's Momentum

Same as the convex version (GeO-N), we can integrate Nesterov's lookahead gradient into SGeO by replacing line 5 with

$$g_t = \nabla f(\theta_t + \mu d_t) \tag{3.8}$$

which we call SGeO-N:

Algorithm 4: SGeO-N (NON-CONVEX)

Initialize θ_1
Set $d_1 = \nabla f(\theta_1)$
for $t = 1$ ***to*** T **do**
    Draw minibatch from training set
    Calculate the gradient $g_t = \nabla f(\theta_t + \mu d_t)$
    Calculate adaptive coefficient $\gamma_t^{NC} = 1 + \bar{g}_t \cdot \bar{d}_t$
    Calculate update $d_{t+1} = \epsilon_t \gamma_t^{NC} \bar{d}_t - \epsilon_t \bar{g}_t$
    Update parameters $\theta_{t+1} = \theta_t + d_{t+1}$

Chapter 4

Experiments

We evaluated SGeO on strongly convex functions with Lipschitz gradients, benchmark deep autoencoder problems and two global optimization benchmarks and compared with various established baselines.

4.1 Baselines

Our baselines are Gradient Descent (GD), Heavy Ball (HB) [8], Nesterov [9], Fletcher-Reeves (FR) [25] and K-FAC. Note that we will use GD, HB, Nesterov and FR for our deterministic experiments which consist of strongly-convex functions with Lipschitz gradients. Moreover, we use HB, Nesterov and K-FAC in our stochastic experiments, i.e. deep autoencoder optimization benchmarks. All the baselines were implemented from scratch except for K-FAC where we used the official code provided by the authors.

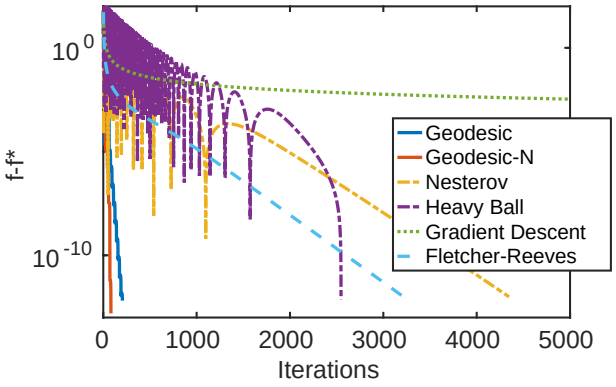
4.2 Strongly Convex Functions with Lipschitz gradients

We borrow three deterministic optimization problems from [13]. The problems are Anisotropic Bowl, Ridge Regression and Smooth-BPDN.

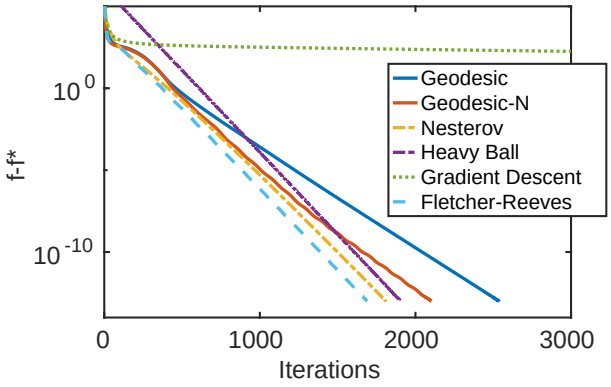
4.2.1 Anisotropic Bowl

The Anisotropic Bowl is a bowl-shaped function with a constraint to get Lipschitz continuous gradients:

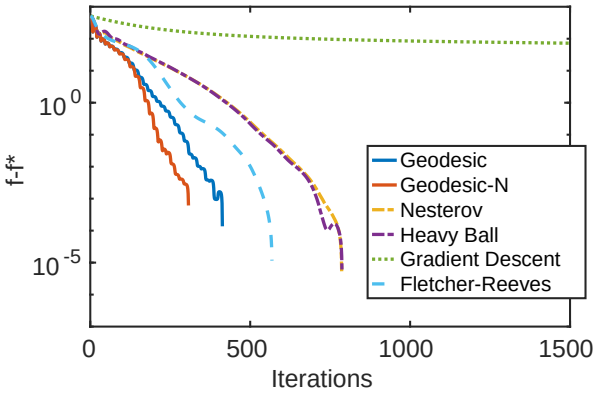
$$\begin{aligned} f(\theta) &= \sum_{i=1}^{n} i \cdot \theta_{(i)}^4 + \frac{1}{2}\|\theta\|_2^2 \\ \text{subject to } &\|\theta\|_2 \leq \tau \end{aligned} \tag{4.1}$$



(a) Anisotropic Bowl



(b) Ridge Regression



(c) Smooth-BPDN

Figure 4.1: Results from experiments on strongly convex functions with Lipschitz gradients. Geodesic and Geodesic-N are our methods and the baselines are Gradient Descent, Heavy Ball, Nesterov and Fletcher-Reeves. All methods start from the same randomly chosen point and are terminated within a tolerance of the global optimum. The horizontal axes show the iterations and the vertical axes show the distance to the optimal value. Figures are best viewed in color.

As in [13], we set $n = 500$, $\tau = 4$ and $\theta_0 = \frac{\tau}{\sqrt{n}}\mathbf{1}$. Thus the Lipschitz and strong convexity parameter will be:

$$L = 12n\tau^2 + 1 = 96001; \quad \ddot{\mu} = 1. \tag{4.2}$$

Figure 4.1a shows the convergence results for our algorithms and the baselines. The algorithms terminate when $f(\theta) - f^* < 10^{-12}$. GeO-N and GeO take only 82 and 205 iterations to converge, while the closest result is that of Heavy-Ball and Fletcher-Reeves which take approximately 2500 and 3000 iterations respectively.

4.2.2 Ridge Regression

The Ridge Regression problem is a linear least squares function with Tikhonov regularization:

$$f(\theta) = \frac{1}{2}\|A\theta - b\|_2^2 + \frac{\lambda}{2}\|\theta\|_2^2 \tag{4.3}$$

where $A \in \mathbb{R}^{m\times n}$ is a measurement matrix, $b \in \mathbb{R}^m$ is the response vector and $\lambda > 0$ is the ridge parameter. The function $f(\theta)$ is a positive definite quadratic function with the unique solution of

$$\theta^* = (A^TA + \lambda I)^{-1}A^Tb \tag{4.4}$$

along with Lipschitz parameter $L = \|A\|_2^2 + \lambda$ and strong convexity parameter $\ddot{\mu} = \lambda$.

Following [13], $m = 1200$, $n = 2000$ and $\lambda = 1$. A is generated from $U\Sigma V^T$ where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times m}$ are random orthonormal matrices and $\Sigma \in \mathbb{R}^{m\times m}$ is diagonal with entries linearly distanced in $[100, 1]$ while $b = \text{randn}(m, 1)$ is drawn (i.i.d) from the standard normal distribution. Thus $\ddot{\mu} = 1$ and $L \approx 1001$.

Figure 4.1b shows the results where Fletcher-Reeves, which is a conjugate gradient algorithm, performs better than other methods but we observe similar performances overall except for gradient descent. The tolerance is set to $f(\theta) - f^* < 10^{-13}$.

4.2.3 Smooth-BPDN

Smooth-BPDN is a smooth and strongly convex version of the BPDN (basis pursuit denoising) problem:

$$\begin{aligned} f(\theta) &= \frac{1}{2}\|A\theta - b\|_2^2 + \lambda\|\theta\|_{\ell_1,\tau} + \frac{\rho}{2}\|\theta\|_2^2 \\ \text{where } \|\theta\|_{\ell_1,\tau} &= \begin{cases} |\theta| - \frac{\tau}{2} & \text{if } |\theta| \geq \tau \\ \frac{1}{2\tau}\theta^2 & \text{if } |\theta| < \tau \end{cases} \end{aligned} \tag{4.5}$$

and $\|\cdot\|_{\ell_1,\tau}$ is a smoothed version of the ℓ_1 norm also known as Huber penalty function with half-width of τ. The function is strongly convex $\ddot{\mu} = \rho$ because of the quadratic term $\frac{\rho}{2}\|\theta\|_2^2$ with Lipschitz constant $L = \|A\|_2^2 + \frac{\lambda}{\tau} + \rho$.

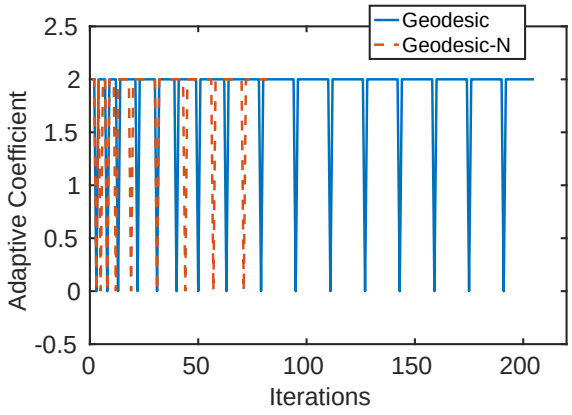
As in [13], we set $A = \frac{1}{\sqrt{n}} \cdot \text{randn}(m, 1)$ where $m = 800$, $n = 2000$, $\lambda = 0.05$, $\tau = 0.0001$ and $\ddot{\mu} = 0.05$. The real signal is a random vector with 40 nonzeros and $b = A\theta^* + e$ where $e = 0.01\frac{\|b\|_2}{\sqrt{m}} \cdot \text{randn}(m, 1)$ is Gaussian noise. Also

$$L = \|A\|_2^2 + \frac{\lambda}{\tau} + \rho \approx 502.7 \tag{4.6}$$

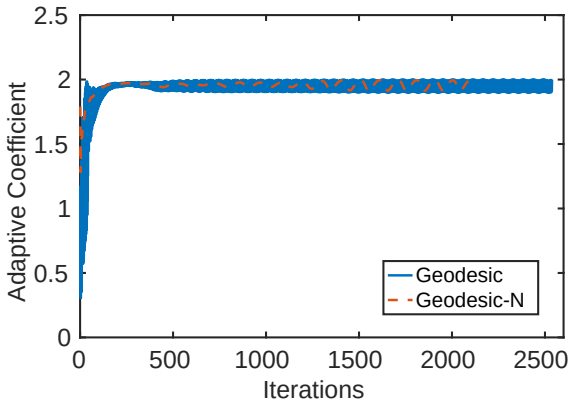
Since we cannot find the solution analytically, Nesterov's solution is used as an approximation to the solution (f_N^*) and the tolerance is set to $f(\theta) - f_N^* < 10^{-12}$. Figure 4.1c shows the results for the algorithms. GeO-N and GeO converge in 308 and 414 iterations respectively, outperforming all other methods. Closest to these two is Fletcher-Reeves with 569 iterations and Nesterov and Heavy Ball converge similarly in 788 iterations.

4.2.4 Geodesic Coefficient Behaviour

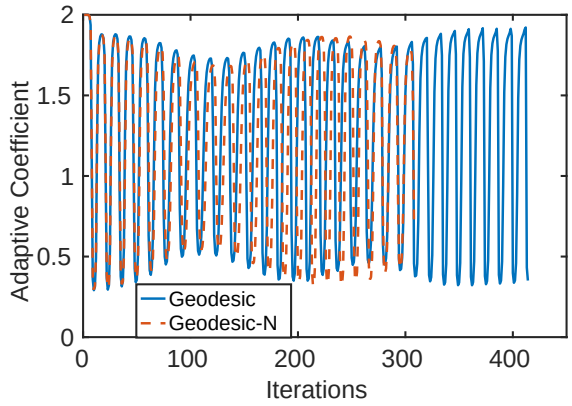
Here we include the values of the adaptive coefficient during optimization from our experiments. The plots in Figures 4.2a to 4.2c show γ_t at each iteration for GeO and GeO-N. For the



(a) Anisotropic Bowl



(b) Ridge Regression



(c) Smooth-BPDN

Figure 4.2: Value of the adaptive coefficient per iteration during training for the strongly convex functions with Lipschitz gradients from our experiments. Note that here $\gamma_t^C = 1 - \bar{g}_t \cdot \bar{d}_t$. Figures are best viewed in color.

Anisotropic Bowl and the Smooth BPDN problem, the values fluctuates between 0 and 2 periodically, with the Smooth-BPDN behaving more sinusoidal and the Anisotropic Bowl more pulse-like. However, for the ridge regression problem, the values gradually increase from 0 in the beginning and stay close to 2 thereafter, indicating $\phi \approx 0$ (convergence) since $\gamma^C = 1 - \cos(\pi - \phi)$.

4.2.5 Details

The learning rate ϵ for all methods is set to $\frac{1}{L}$ except for Nesterov which is set to $\frac{4}{3L+\bar{\mu}}$ (optimal learning rate). The momentum parameter μ for Heavy Ball, Nesterov and GeO-N is set to the following:

$$\mu = \frac{1 - \sqrt{\bar{\mu}\epsilon}}{1 + \sqrt{\bar{\mu}\epsilon}}$$

where L is the Lipschitz parameter and $\ddot{\mu}$ is the strong-convexity parameter. The adaptive parameter γ_t for Fletcher-Reeves is set to γ_t^{FR} and for GeO and GeO-N is $\gamma_t^C = 1 - \bar{g}_t \cdot \bar{d}_t$. The function-specific parameter α is set to 1, 0.5 and 0.9 in that order for the mentioned problems. It's important to note that the approximate conjugate gradient method is only exact when an exact line search is used, which is not the case in our experiments with a quadratic function (Ridge Regression).

4.3 Deep Autoencoders

To evaluate the performance of SGeO on non-convex problems, we apply it to 3 benchmark deep autoencoder problems first introduced in [27] which use three datasets, MNIST, FACES and CURVES. Due to the difficulty of training these networks, they have become standard benchmarks for neural network optimization [27, 2, 5].

4.3.1 Datasets

MNIST MNIST [28] is a dataset of handwritten digits containing 70000 gray-scale images of which 60000 were used for training and 10000 is used for testing. Each image is 128×128 pixels of values between 0 and 255. The image vectors were normalized by one prior to training (and testing) following the literature [27]. Samples of the dataset can be found in Figure 4.3a.

FACES The Olivetti Faces dataset (taken from the Olivetti database at AT&T) is a dataset of 400 gray-scale 64×64 faces of 40 different people. The dataset was expanded to contain 165600 images of size 25×25 through various transformations [27] which was then split into 103500 training and 41400 testing images. Samples of the dataset can be found in Figure 4.3b.

CURVES The Curves [27] dataset consists of 30000 images of synthetic 28×28 random gray-scale curves where it's split into 20000 training and 10000 testing images. Samples of the dataset can be found in Figure 4.3c.

4.3.2 Details

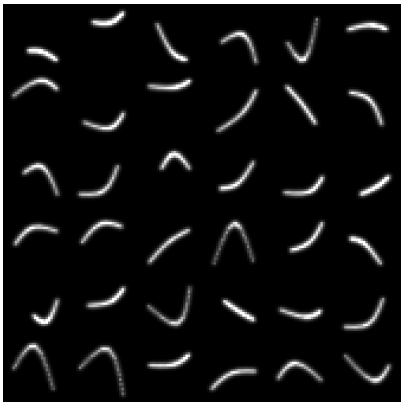
Loss Function All three autoencoder experiments attempt to minimize the cross-entropy loss:

$$-\sum_i X_i \log(\hat{X}_i) - \sum_i (1 - X_i) \log(1 - \hat{X}_i) \tag{4.7}$$

where X_i is the input vector and $\hat{X}_i$ is the predicted output vector. The gradients are calculated via back-propagation [29]. However, note that we report the reconstruction error

(a) MNIST dataset.

(b) FACES dataset.



(c) CURVES dataset

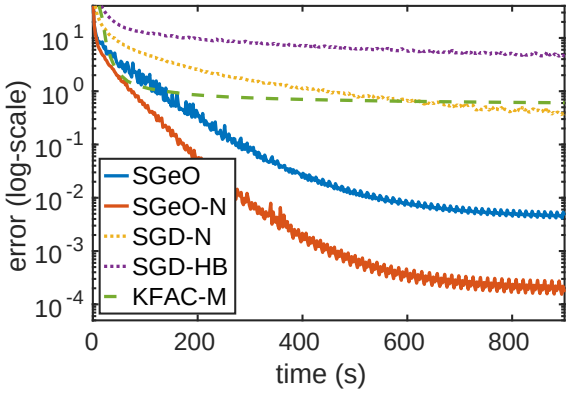
Figure 4.3: Samples taken from the three datasets used in deep autoencoder experiments. Note that the Faces dataset used here is taken from [27] whose images have undergone several transformations which include rotation, cropping, scaling and subsampling the original Olivetti dataset.

in our experiments:

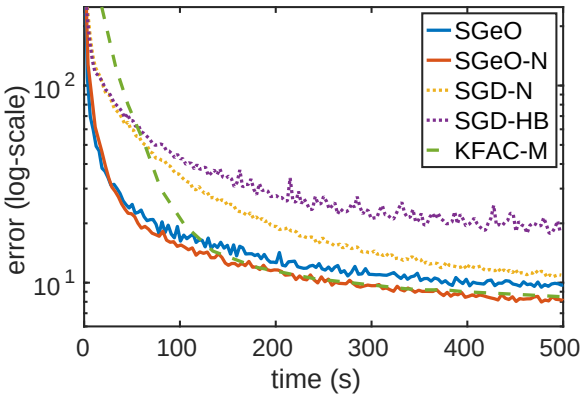
$$\sum_i (X_i - \hat{X}_i)^2 \tag{4.8}$$

Architectures To be consistent with previous literature [3, 2, 5], we use the same network architectures as in [27] and also report the reconstruction error instead of the log-likelihood objective. The layer structure for MNIST, FACES and CURVES are [1000 500 250 30 250 500 1000], [2000 1000 500 30 500 1000 2000] and [400 200 100 50 25 6 25 50 100 200 400] respectively. All layers use sigmoid activations except the middle layer for MNIST, the middle and the last layer for FACES and the middle layer for CURVES which use linear activation.

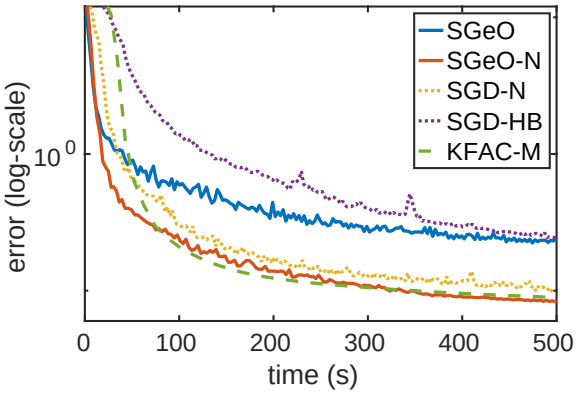
Implementation Both the baselines and our algorithms were implemented using MATLAB with single precision on a single machine with a 3.6 GHz Intel CPU and an NVIDIA GeForce



(a) MNIST dataset.



(b) FACES dataset.

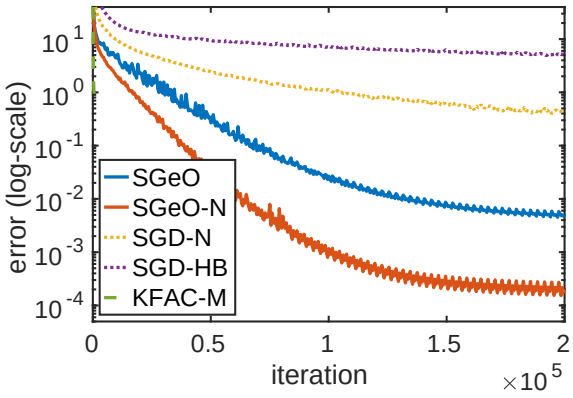


(c) CURVES dataset

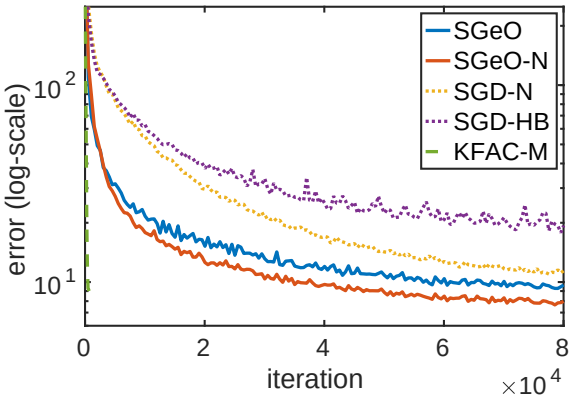
Figure 4.4: Results from autoencoder experiments. The horizontal axis shows computation time and the vertical axis shows log-scale training error. Our methods are SGeO and SGeO-N and the baselines are SGD-HB and SGD-N, variants of SGD that use the Heavy Ball momentum and Nesterov's momentum respectively, along with K-FAC. All methods use the same initialization. SGeO-N is able to outperform other methods on the MNIST dataset and performs similarly to K-FAC on the other two while outperforming other baselines. Figures are best viewed in color.

GTX 1080 Ti GPU with 11 GBs of memory. We also implemented the autoencoders and their corresponding back-propagation from scratch in MATLAB to increase efficiency.

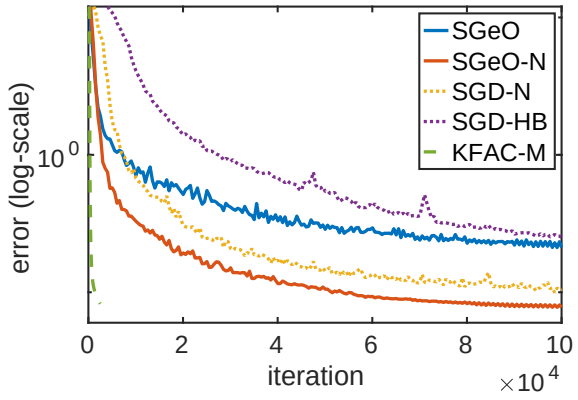
Results The results are shown in Figures 4.4a to 4.4c. Since we are mainly interested in optimization and not in generalization, we only report the training error, although we include the test set performances in the next subsection. We report the reconstruction relative to the computation time to be able to compare with K-FAC, since each iteration of K-FAC takes orders of magnitude longer than SGD and SGeO. We also include the per-iteration results for the autoencoder experiments in Figures 4.5a to 4.5c. We reported the reconstruction error vs. running time to make it easier to compare to K-FAC. K-FAC, which is a second-order algorithm, converges in fewer iterations but has a high per-iteration cost. All other methods are first-order and have similar per-iteration costs.



(a) MNIST dataset.



(b) FACES dataset.



(c) CURVES dataset

Figure 4.5: Results from autoencoder experiments on three datasets. The horizontal axis shows iterations and the vertical axis shows log-scale training error. K-FAC is a second-order method and takes much fewer iterations to converge. However, the per-iteration cost is much higher than the other methods. SGeO and SGeO-N are our methods and the baselines are Nesterov (SGD-N), Heavy Ball (SGD-HB) and KFAC-M (M indicating a form of momentum). Figures are best viewed in color.

Parameters All methods use the same parameter initialization scheme known as "sparse initialization" introduced in [3]. This initialization scheme sets a limited number of randomly chosen incoming weights (here, 15) to a node, to a small random number and sets the others to zero. The experiments for the Heavy Ball algorithm and SGD with Nesterov's momentum follow [2] which were tuned to maximize performance for these problems. For SGeO, we chose a fixed momentum parameter and used a simple multiplicative schedule for the learning rate:

$$\epsilon_t = \epsilon_1 \times \beta_\epsilon^{\lfloor \frac{t}{K} \rfloor}$$

where the initial value (ϵ_1) was chosen from {0.1,0.15,0.2,0.3,0.4,0.5} and is decayed (K) every 2000 iterations (parameter updates). The decay parameter (β_ϵ) was set to 0.95. For the momentum parameter μ, we did a search in {0.999,0.995,0.99}. The minibatch size was set to 500 for all methods except K-FAC which uses an exponentially increasing schedule for the

minibatch size. For K-FAC we used the official code provided [1] by the authors with default parameters to reproduce the results. The version of K-FAC we ran was the Blk-Tri-Diag approach which achieves the best results in all three cases. To do a fair comparison with other methods, we disabled iterate averaging for K-FAC. It is also worth noting that K-FAC uses a form of momentum [5].

Comparison In all three experiments, SGeO-N is able to outperform the baselines (in terms of reconstruction error) and performs similarly as (if not better than) K-FAC. We can see the effect of the adaptive coefficient on the Heavy Ball method, i.e. SGeO, which also outperforms SGD with Nesterov's momentum in two of the experiments, MNIST and FACES, and also outperforms K-FAC in the MNIST experiment. Use of Nesterov style lookahead gradient significantly accelerates training for the MNIST and CURVES dataset, while we see this to a lesser extent in the FACES dataset. This is also the case for the other baselines [2, 5]. Further, we notice an interesting phenomena for the MNIST dataset (Figure 4.4a). Both SGeO and SGeO-N reach very low error rates, after only 900 seconds of training, SGeO and SGeO-N arrive at an error of 0.004 and 0.0002 respectively.

4.3.3 Generalization Experiments

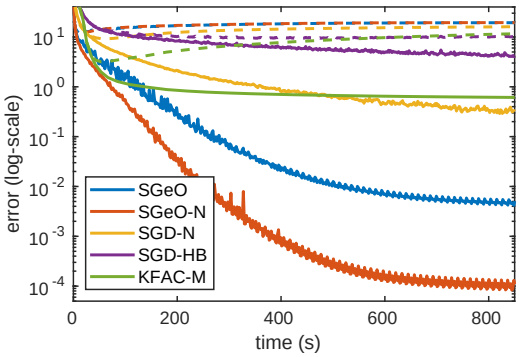
We include generalization experiments on the test set here. However, as mentioned before, our focus is optimization and not generalization, we are aware that the choice of optimizer can have a significant effect on the performance of a trained model in practise. Results are shown in Figures 4.6a to 4.6c. SGeO-N shows a significant better predictive performance than SGeO on the CURVES data set and both perform similarly on the two other datasets. On MNIST and FACES (Figures 4.6a and 4.6b) all methods generalize similarly, overfitting the training data. On the CURVES dataset however, K-FAC performs better than others while Nesterov style methods (SGD-N and SGeO-N) deliver the same performance and SGD-HB and SGeO reach higher errors on the testing set.

Note that the algorithms are tuned for best performance on the training set. Overfitting can be dealt with in various ways such as using appropriate regularization during training, using a small validation set to tune the parameters and early stopping. One can also use more modern regularization techniques such as Dropout [30]. Dropout works by randomly dropping (deleting) a portion of the nodes during training and using all nodes during testing. It can reinterpreted as using an ensemble of "thinner" neural networks where the result is the average of the ensemble. We leave this for future work.

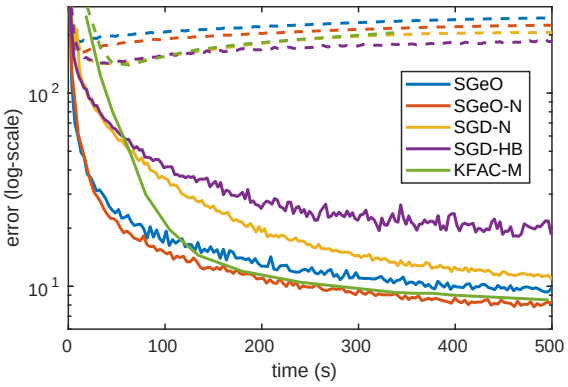
4.3.4 Geodesic Coefficient Behaviour

The adaptive coefficient values during training for the autoencoder experiments is shown in Figures 4.7a to 4.7c. The values for all three problem stay close to values between 1 and 1.5 indicating $\frac{\pi}{2} < \phi < \frac{2\pi}{3}$. However, interpreting these values would not be as exact as our

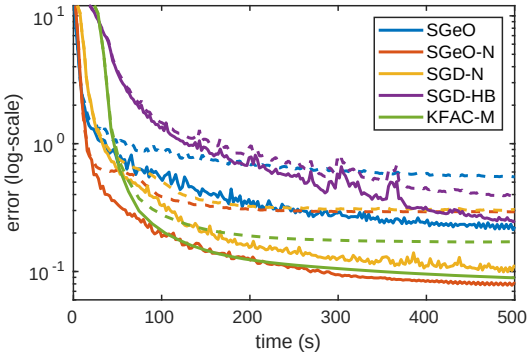
[1] http://www.cs.toronto.edu/ jmartens/docs/KFAC3-MATLAB.zip



(a) MNIST dataset.



(b) FACES dataset.



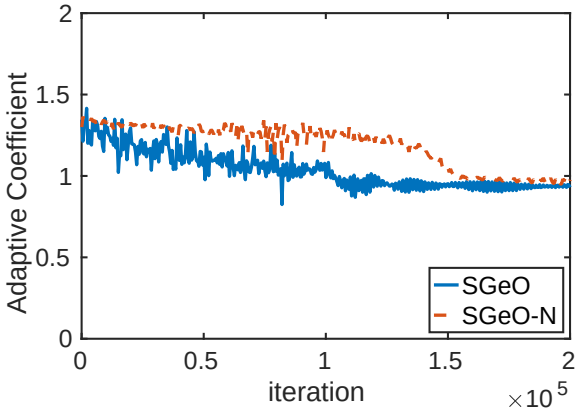
(c) CURVES dataset.

Figure 4.6: Performance of the autoencoders evaluated on the test set while training. Note that in all cases the algorithms are tuned to maximize performance on the training set. The dashed lines show the test error while the continuous lines show the training error. The letter "N" in SGeO-N indicates the use of Nesterov style lookahead gradient. Figures are best viewed in color.

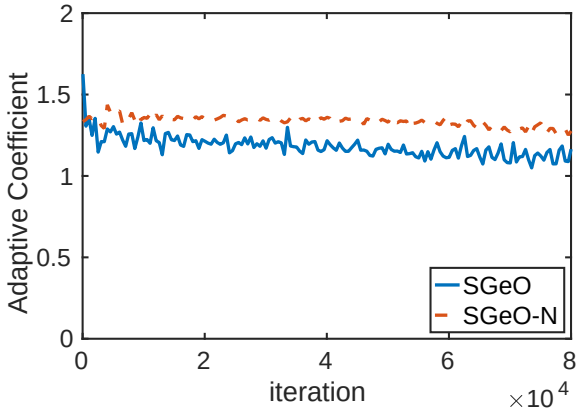
convex experiments due to partial gradient information, stochasticity and the non-convex nature of the problems. Further, adding lookahead gradient clearly decreases fluctuations in the adaptive coefficient value, thus a more stable training process.

4.4 Global Optimization Benchmarks

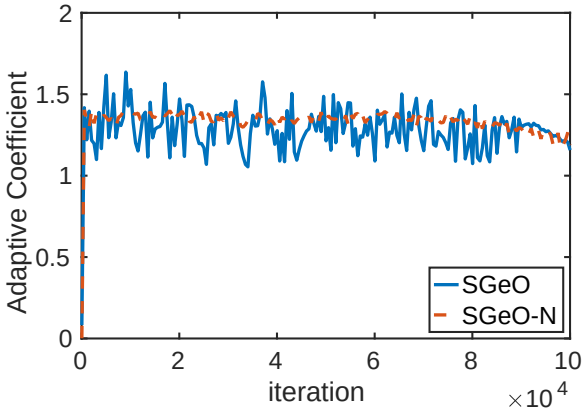
We compared our method (non-stochastic version using γ^{NC}) with Nesterov's method on global optimization benchmarks with two dimensions to facilitate visualization. The results showing the contour plots and the paths taken by the algorithms can be found in Figure 4.8.



(a) MNIST dataset.



(b) FACES dataset.



(c) CURVES dataset

Figure 4.7: Behaviour of the adaptive coefficient per iteration for the autoencoder experiments. Note that here $\gamma_t^{NC} = 1 + \bar{g}_t \cdot \bar{d}_t$ and the experiments are taken from optimization on the training set. We can see the effect of Nesterov's lookahead gradient on the autoencoder benchmarks. Figures are best viewed in color.

4.4.1 Levy Function

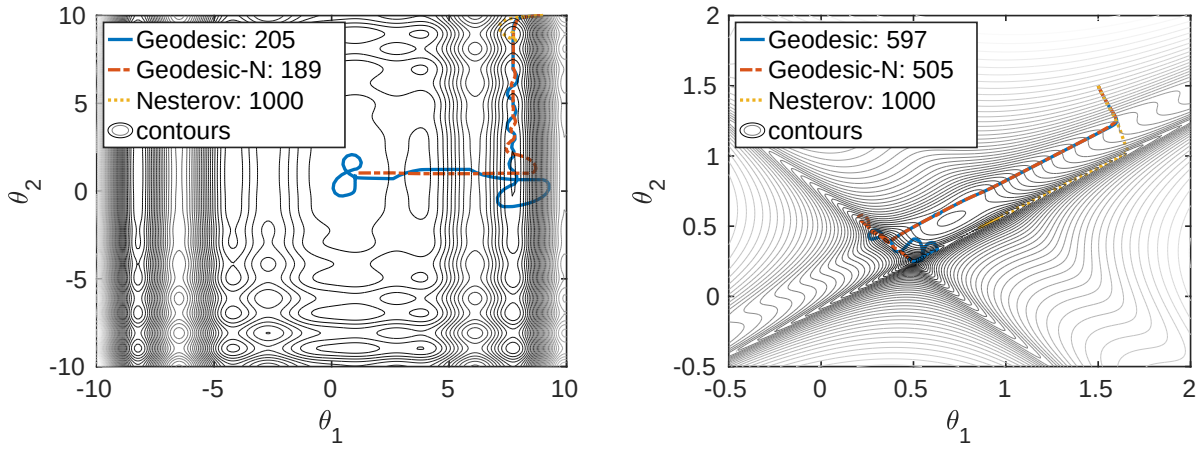
The Levy function [31, 32] features a wavy surface in both directions, multiple ravines and local . The global minimum is at $(0, 0)$. The function is:

$$f(\theta) = \sin^2(\pi w_1) + (w_1 - 1)^2[1 + 10\sin^2(\pi w_1 + 1)] + (w_2 - 1)^2[1 + \sin^2(2\pi w_2)] \quad (4.9)$$

where $w_i = 1 + \frac{\theta_i - 1}{4}$ for $i = 1, 2$. We initialize all three methods at $(9, 10)$, which was experimentally chosen as a suitable starting point.

4.4.2 Scaled Goldstein-Price Function

The scaled Godstein-Price function [31, 33] features several local minima, ravines and plateaus which can be representative of a deep neural network's loss function. The global minimum is



(a) Levy function.

(b) Goldstein-Price function.

Figure 4.8: Results from the scaled Goldstein-Price and the Levy function experiments. The figures show the contour plots of the functions and the paths taken by the methods in different colors. The numbers in front of each method's name in the legend show the total iterations it took for the algorithm to get to a tolerance of 0.001 from the global minimum or reached maximum iterations (1000). Geodesic is our method and N indicates the use of Nesterov's lookahead gradient. The global minimum for (a) is at (0,0) and for (b) is at (0.5,0.25). All methods start from the same point and the hyper-parameters are tuned for best performance. Although the algorithms used are not stochastic, the adaptive coefficient used for these problems is γ^{NC} since the functions are non-convex. Figures are best viewed in color.

at $(0.5, 0.25)$. The function is:

$$f(\theta) = \frac{1}{2.427}\Big[\log\Big([1+(\bar{\theta}_1+\bar{\theta}_2+1)^2(19-14\bar{\theta}_1+3\bar{\theta}_1^2-14\bar{\theta}_2+6\bar{\theta}_1\bar{\theta}_2+3\bar{\theta}_2^2)] \\ [30+(2\bar{\theta}_1-3\bar{\theta}_2)^2(18-32\bar{\theta}_1+12\bar{\theta}_1^2+48\bar{\theta}_2-36\bar{\theta}_1\bar{\theta}_2+27\bar{\theta}_2^2)]\Big)-0.8693\Big] \tag{4.10}$$

where $\bar{\theta}_i = 4\bar{\theta}_i - 2$ for $i = 1, 2$. We initialize all methods at $(1.5, 1.5)$, which was experimentally chosen as a suitable starting point.

Details The momentum parameter μ for both Nesterov and Geodesic-N was set to 0.9. The learning rate for all methods is fixed and is tuned for best performance. The results from both experiments indicate that Geodesic is able to effectively escape local minima and recover from basins of attraction, while Nesterov's method gets stuck at local minima in both cases. We can also observe the effect of lookahead gradient on our method where the path taken by Geodesic-N is much smoother than Geodesic.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

We proposed two novel and efficient algorithms based on adaptive coefficients for the Heavy Ball method inspired by a geodesics. We compared our algorithms against Gradient Descent with Nesterov's Momentum and regular momentum (Heavy Ball) and a recently proposed second-order method, K-FAC, on three strongly convex functions with Lipschitz gradients and three deep autoencoder optimization benchmarks. We saw that the algorithms are able to outperform all first-order methods that we compared to, by a notable margin. We called our algorithms GeO and GeO-N (also SGeO and SGeO-N for the stochastic case) and analyzed the geodesic coefficient behaviour in different situations. Our generalization experiments show that both the baselines and our algorithms overfit to the training data, however, we also discussed methods to overcome overfitting such as using a small validation set. We also visualized the algorithms' paths on two global optimization benchmarks.

Our proposed algorithms are easy to implement and their computational overhead over first-order methods, i.e. calculating the dot product, is marginal. They can also perform as effectively as or better than second-order methods (here, K-FAC) without the need for expensive higher-order operations in terms of computation time (and memory). We believe that geodesic optimization opens new and promising directions in high-dimensional neural network optimization and is a step towards combining first-order and second-order methods.

5.2 Future Work

For future work, we are planning to apply our adaptive momentum algorithm to other machine learning paradigms such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs) and Reinforcement Learning algorithms such as Policy Gradients. It also remains to analyse the theoretical properties of the algorithms such as their convergence rate in convex cases with various constraints such as smoothness, strong-convexity and Lipschitzness which we leave for future work.

Bibliography

[1] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[2] Ilya Sutskever et al. "On the importance of initialization and momentum in deep learning." In: *ICML (3)* 28.1139-1147 (2013), p. 5.

[3] James Martens. "Deep learning via Hessian-free optimization." In: *ICML*. Vol. 27. 2010, pp. 735–742.

[4] Shun-Ichi Amari. "Natural gradient works efficiently in learning". In: *Neural computation* 10.2 (1998), pp. 251–276.

[5] James Martens and Roger Grosse. "Optimizing neural networks with kronecker-factored approximate curvature". In: *International conference on machine learning*. 2015, pp. 2408–2417.

[6] Jean François Aujol et al. "Optimal convergence rates for Nesterov acceleration". In: *arXiv preprint arXiv:1805.05719* (2018).

[7] Ricky Fok, Aijun An, and Xiaogong Wang. "Geodesic and contour optimization using conformal mapping". In: *Journal of Global Optimization* 69.1 (2017), pp. 23–44.

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

[9] Yurii E Nesterov. "A method for solving the convex programming problem with convergence rate O (1/kˆ 2)". In: *Dokl. akad. nauk Sssr*. Vol. 269. 1983, pp. 543–547.

[10] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), p. 436.

[11] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[12] Léon Bottou, Frank E Curtis, and Jorge Nocedal. "Optimization methods for large-scale machine learning". In: *Siam Review* 60.2 (2018), pp. 223–311.

[13] Xiangrui Meng and Hao Chen. "Accelerating Nesterov's method for strongly convex functions with Lipschitz gradient". In: *arXiv preprint arXiv:1109.6058* (2011).

[14] Weijie Su, Stephen Boyd, and Emmanuel Candes. "A differential equation for modeling Nesterov's accelerated gradient method: Theory and insights". In: *Advances in Neural Information Processing Systems*. 2014, pp. 2510–2518.

[15] Andre Wibisono, Ashia C Wilson, and Michael I Jordan. "A variational perspective on accelerated methods in optimization". In: *proceedings of the National Academy of Sciences* 113.47 (2016), E7351–E7358.

[16] Aaron Defazio. "On the Curved Geometry of Accelerated Optimization". In: *arXiv preprint arXiv:1812.04634* (2018).

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

[18] Matthew D Zeiler. "ADADELTA: an adaptive learning rate method". In: *arXiv preprint arXiv:1212.5701* (2012).

[19] Tijmen Tieleman and Geoffrey Hinton. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude". In: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31.

[20] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[21] Brendan O'donoghue and Emmanuel Candes. "Adaptive restart for accelerated gradient schemes". In: *Foundations of computational mathematics* 15.3 (2015), pp. 715–732.

[22] James Lucas et al. "Aggregated momentum: Stability through passive damping". In: *arXiv preprint arXiv:1804.00325* (2018).

[23] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. "On the convergence of adam and beyond". In: (2018).

[24] Magnus Rudolph Hestenes and Eduard Stiefel. *Methods of conjugate gradients for solving linear systems*. Vol. 49. 1. NBS Washington, DC, 1952.

[25] Reeves Fletcher and Colin M Reeves. "Function minimization by conjugate gradients". In: *The computer journal* 7.2 (1964), pp. 149–154.

[26] Yvonne Choquet-Bruhat. *Introduction to General Relativity, Black Holes, and Cosmology*. OUP Oxford, 2014.

[27] Geoffrey E Hinton and Ruslan R Salakhutdinov. "Reducing the dimensionality of data with neural networks". In: *science* 313.5786 (2006), pp. 504–507.

[28] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[29] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[30] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. URL: `http://jmlr.org/papers/v15/srivastava14a.html`.

[31] S. Surjanovic and D. Bingham. *Virtual Library of Simulation Experiments: Test Functions and Datasets*. Retrieved April 14, 2019, from `http://www.sfu.ca/~ssurjano`.

[32] Manuel Laguna and Rafael Martı. "Experimental testing of advanced scatter search designs for global optimization of multimodal functions". In: *Journal of Global Optimization* 33.2 (2005), pp. 235–255.

[33] Victor Picheny, Tobias Wagner, and David Ginsbourger. "A benchmark of kriging-based infill criteria for noisy optimization". In: *Structural and Multidisciplinary Optimization* 48.3 (2013), pp. 607–626.