Learning how to learn: Descending down Steep Curves

A B. Tech Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Amish Mittal
(1801CS07)

under the guidance of

Prof Jimson Mathew

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY PATNA
PATNA - 800013, BIHAR

CERTIFICATE

*This is to certify that the work contained in this thesis entitled "**Learning how to learn: Descending down Steep Curves**" is a bonafide work of **Amish Mittal (Roll No. 1801CS07)**, carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Patna under my supervision and that it has not been submitted elsewhere for a degree.*

Supervisor: **Prof Jimson Mathew**

Professor,

May, 2022

Department of Computer Science & Engineering,

Patna.

Indian Institute of Technology Patna, Bihar.

Acknowledgements

To my supervisor and mentor Prof Jimson for always supporting me in my work and helping me achieve my career goals.

To Abhinav for helping me with a lot of ideas included here.

To Abhishek, Shashwat, Kunj, Hrishabh, Vanshika, Shreya and Vijaya for making the last four years memorable. To Aditi for restoring some happiness in my life. To Lakshya for always being there.

To my parents and Sana for loving me all they could.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Gradient Descent

An Artificial Neural Network (ANN) can be written as $Y = h(\theta, X)$.

The goal of an optimization problem over such a network is to approximate correct value of Y by fixing h and then using a loss function f to estimate the parameters θ.

Ideally we want that the generalization risk $f(\theta) = \mathbb{E}_{(x,y)\sim p_{\text{data}}} L(h(\theta, X), Y)$ is minimized where p is the data generating distribution. But as p is not available to us, we optimize $f^*(\theta) = \mathbb{E}_{(x,y)\sim \tilde{p}_{\text{data}}} L(h(\theta, X), Y)$, where $\tilde{p}$ is the empirical distribution available with us. We hope that by minimizing f^*, we would also minimize f.

We now minimize the empirical risk which can be written as:

$$f^*(\theta) = \mathbb{E}_{(x,y)\sim \tilde{p}_{\text{data}}} L(h(\theta, X), Y) = \frac{1}{m} \sum_i L(h(\theta, x^i), y^i)$$

Gradient descent is one of the approaches to update the parameters of the network. It is an iterative first-order optimisation algorithm used to find a local minimum/maximum of a given function. This method is commonly used in machine learning (ML) and deep

learning(DL) to minimise a cost/loss function. The iterative parameter update is defined as:

$$\theta_t = \theta_{t-1} - \eta \ \nabla_\theta \ f^*(\theta)$$

1.2 Diverging or Overshooting the minima

A common problem with gradient descent based algorithms is that they often diverge or overshoot the local or global minima, which makes their convergence impossible. This is frequent for non-convex loss functions but can occur in convex loss functions also. This can occur when $\nabla \ f- > \infty$ due to extremely high gradient value which makes $\eta \ \nabla \ f- > \infty$. Thus, the optimization diverges. In case the curve becomes very steep, it is not able to "descend down" and becomes highly susceptible to the chosen learning rate η which we can't determine beforehand. This also leads to a non-deterministic pattern of the descent, which may not be desirable. This dilemma or choosing a high or a low learning rate is summarized in Figure 1.1.

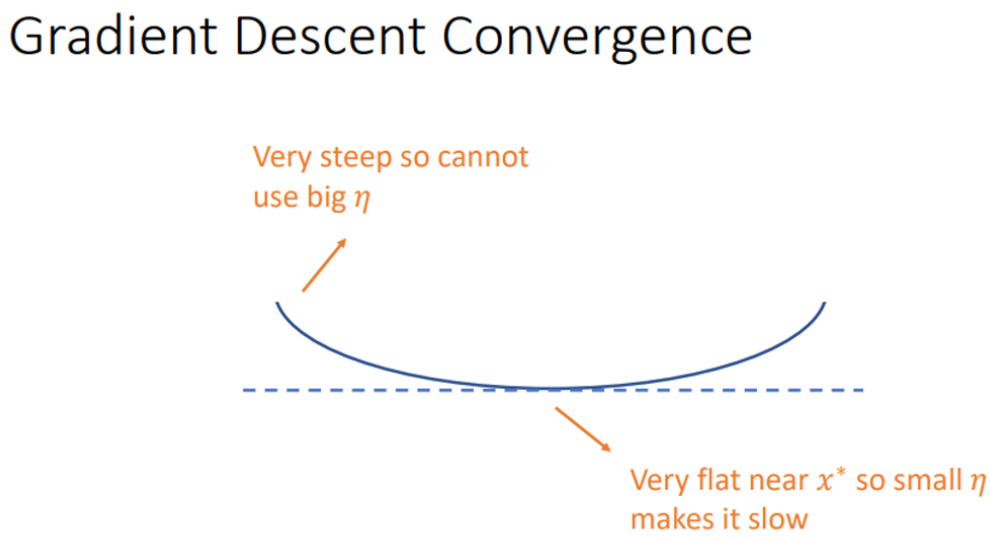


Fig. 1.1 Dilemma of choosing a high or low learning rate. Source: https://praneethnetrapalli.org/

1.3 Intuition to normalize the descent

We make an attempt to normalize the descent, i.e. make sure that the descent does not diverge and the updates are consistent with each other. For this, we make certain changes to the update rule which are highlighted in Chapter 3. In sense, we make the descent non-monotonic over the l_2 norm of gradient value ($||\nabla_\theta f(\theta)||_2$), i.e. if the norm of gradient is small, the parameters updates should be small; if the norm of gradient is high, then also the parameter updates should be small; and the maximum parameter update would happen when the norm of gradient is some moderate value. We model this based on the projectile motion of a particle when hit from an inclined plane using Newton's Laws of Motion. This is shown in Figure 1.2.

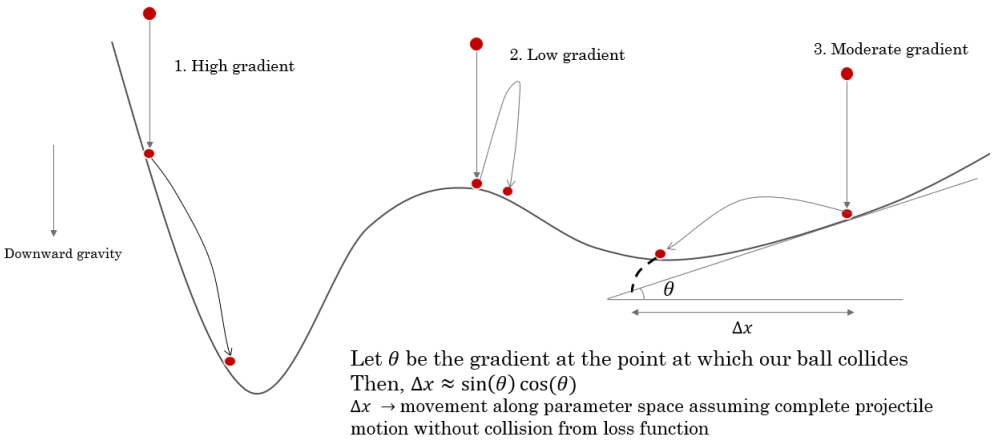


Fig. 1.2 Taking inspiration from projectile motion on an inclined plane.

1.4 Constraints and Goals of this hypothesis

The constraints of forming such our hypothesis lie on our understanding of gradient descent and non-convex optimization. We model the next results based on having a gradient descent optimizer which:

1. Does not perform worse as compared to other optimizers

2. Diverges in minimum number of cases

3. Its convergence and convergence bound can be proved

4. It has a similar asymptotic time complexity as compared to other optimizers

.

We treat the below goals as our best case scenarios:

1. Gives a lower loss value/better convergence value

2. It takes less number of epochs to reach convergence

3. Converges towards global minima instead of local minima

4. Its REGRET [BN12] bound is $O(\sqrt[2]{T})$

Chapter 2

Review of Prior Works

Gradient descent is one of the most popular algorithms to perform optimization and by far the most common way to optimize neural networks. In this section we will look at a few of the famous algorithms (referred to as optimizers henceforth) suggested in various literature. These algorithms, however, are often used as black-box optimizers, as practical explanations of their strengths and weaknesses are hard to come by. [Rud16]

2.1 Gradient Descent Optimizers

2.1.1 Momentum [Qia99]

SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another, which are common around local optima. In these scenarios, SGD oscillates across the slopes of the ravine while only making hesitant progress along the bottom towards the local optimum. Momentum is a method that helps accelerate SGD in the relevant direction and dampens oscillations. It does this by adding a fraction γ of the update vector of the past time step to the current update vector.

$$v_t = \gamma \; v_{t-1} + \eta \; \nabla_\theta \; f(\theta)$$

$$\theta_t = \theta_{t-1} - v_t$$

2.1.2 Adagrad [DHS11]

Adagrad adapts the learning rate to the parameters, performing smaller updates (i.e. low learning rates) for parameters associated with frequently occurring features, and larger updates (i.e. high learning rates) for parameters associated with infrequent features.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

$G_t \in \mathbb{R}^{dxd}$ here is a diagonal matrix where each diagonal element i, i is the sum of the squares of the gradients w.r.t. θ_i up to time step t, while ϵ is a smoothing term that avoids division by zero.

2.1.3 Adadelta [Zei12]

Adadelta is an extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate. Instead of accumulating all past squared gradients, Adadelta restricts the window of accumulated past gradients to some fixed size w. The authors first define another exponentially decaying average, this time not of squared gradients but of squared parameter updates,

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2$$

The root mean squared error of parameter updates is thus:

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon}$$

Since $RMS[\Delta\theta]_t$ is unknown, we approximate it with the RMS of parameter updates until the previous time step. Replacing the learning rate η in the previous update rule with

$RMS[\Delta\theta]_{t-1}$ finally yields the Adadelta update rule:

$$\begin{aligned} \Delta\theta_t &= -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t \\ \theta_{t+1} &= \theta_t + \Delta\theta_t \end{aligned} \tag{2.1}$$

2.1.4 RMSProp

RMSprop is an unpublished, adaptive learning rate method proposed by Geoff Hinton in Lecture 6e of his Coursera Class. [1]

$$\begin{aligned} E[g^2]_t &= 0.9E[g^2]_{t-1} + 0.1g_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \end{aligned} \tag{2.2}$$

RMSprop as well divides the learning rate by an exponentially decaying average of squared gradients. Hinton suggests γ to be set to 0.9, while a good default value for the learning rate η is 0.001.

2.1.5 Adam [KB14]

Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients v_t like RMSprop, Adam also keeps an exponentially decaying average of past gradients m_t, similar to momentum.

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \end{aligned} \tag{2.3}$$

m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively, hence the name of the method. As

[1] http://www.cs.toronto.edu/ tijmen/csc321/slides/lecture_slides_lec6.pdf

m_t and v_t are initialized as vectors of 0's, the authors of Adam observe that they are biased towards zero, especially during the initial time steps, and especially when the decay rates are small. They counteract these biases by computing bias-corrected first and second moment estimates:

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \end{aligned} \tag{2.4}$$

Then they are used in the Adam update rule:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

The authors propose default values of 0.9 for $\beta_1, 0.999$ for b_2, and 10^{-8} for ϵ.

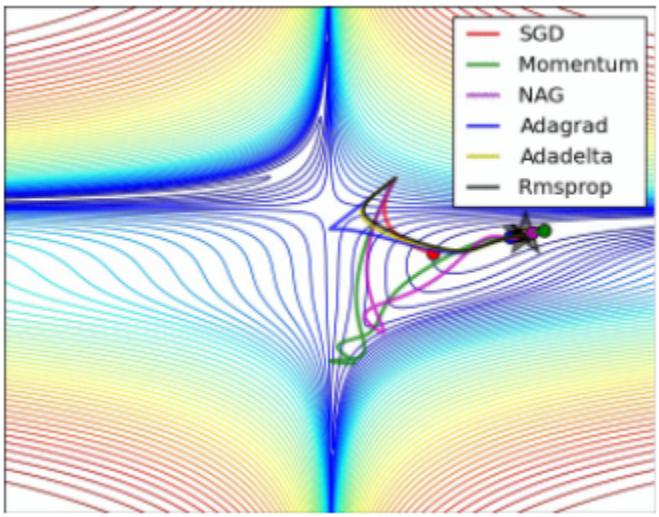


Fig. 2.1 Various Optimizers over Loss contour plot

2.2 Conclusion

This chapter provided details of the some of the popular gradient descent optimizers suggested in various literature. If your input data is sparse, then you likely achieve the best

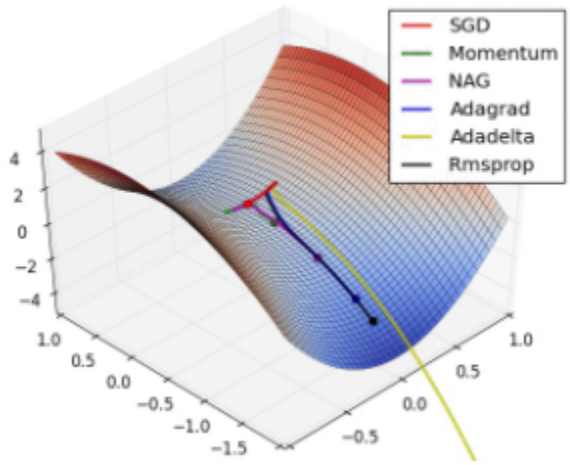


Fig. 2.2 Various Optimizers over a saddle point

results using one of the adaptive learning-rate methods. An additional benefit is that you won't need to tune the learning rate but likely achieve the best results with the default value. In summary, RMSprop is an extension of Adagrad that deals with its radically diminishing learning rates. It is identical to Adadelta, except that Adadelta uses the RMS of parameter updates in the numinator update rule. Adam, finally, adds bias-correction and momentum to RMSprop. Insofar, RMSprop, Adadelta, and Adam are very similar algorithms that do well in similar circumstances. Insofar, Adam might be the best overall choice.

In the next chapter, we discuss our own normalizing factor for these optimizers.

Chapter 3

Normalizing parameter update

3.1 Normalizing factor

We propose a novel gradient descent optimizer which is non-monotonic on the gradient value opposite to other gradient descent optimizers as seen in the Chapter 2. The new update rule on the model parameters $\theta \in \mathbb{R}^n$ is defined as:

$$\Delta\theta_i = -\max\left(\epsilon,\ \frac{|\nabla f_i|^h}{|1+\nabla f_i^2|^h}\right).\nabla f_i \tag{3.1}$$

where, $\Delta\theta_i$ = parameter change along i^{th} dimension,

∇f_i = gradient of loss function f along i^{th} dimension

h = hyperparameter to control convergence

ϵ = small constant $\mathbb{R}$ as smoothning term

3.2 Intuition for this normalization factor

As seen in Figure 1.2, we assume our descent to be analogous to a projectile motion on inclined plane. Using simple Newton's Laws of Motion we can show that the range of the

projectile Δx would approximately be equal to:

$$\Delta x \approx -\sin(\theta)\cos(\theta) \approx -\frac{\nabla f}{1+|\nabla f|^2}$$

Expanding this to $\mathbb{R}^n$ as $x \in \mathbb{R}^n$,

$$\Delta x_i = -\frac{1}{1+|\nabla f_i|^2}.\nabla f_i, \text{ where } \tan(\theta) = \nabla f$$

We now Vectorize this operation to improve the time complexity,

$$\Delta = -\frac{1}{1+F} \odot \nabla f$$

where,

$$F = \begin{bmatrix} \nabla f_0^2 & 0 & 0 & 0 \\ 0 & \nabla f_1^2 & 0 & 0 \\ 0 & 0 & \nabla f_2^2 & 0 \\ 0 & 0 & 0 & \nabla f_3^2 \end{bmatrix}$$

This update normalization factor will make sure that the descent is of the form as shown in Figure 3.1.

3.3 Variations to form update rule

To improve the performance of our optimizer, we add terms to it and make certain modification as inspired from other optimizers, particularly Momentum and RMSProp. The variations are highlighted below:

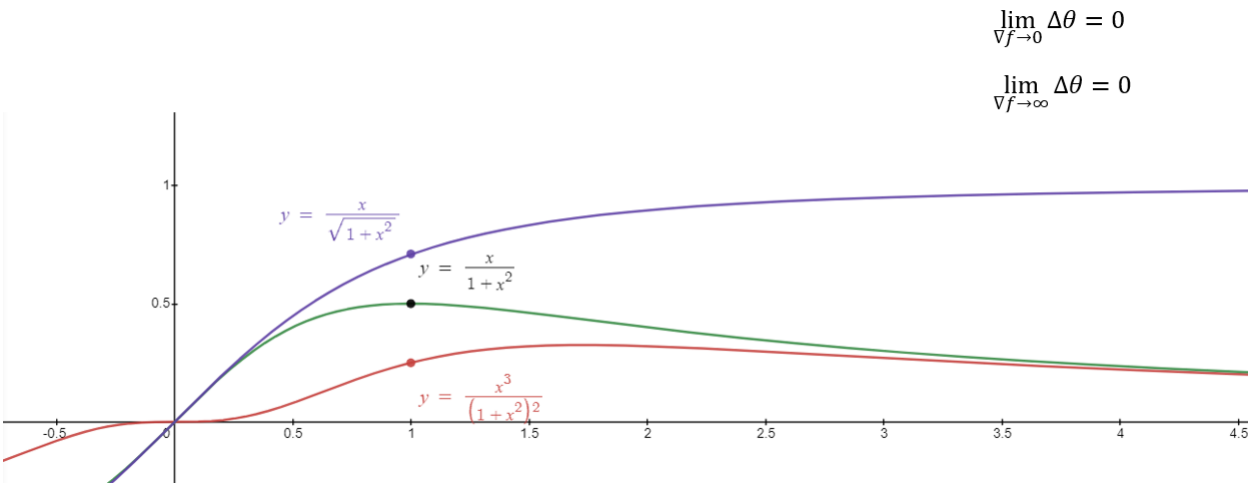


Fig. 3.1 Non-monotonic nature of normalization factor

Vanilla

This has same expression which we have shown above. It is henceforth referred to as "my optimizer".

$$\Delta\theta_i = -\max\left(\epsilon,\ \frac{|\nabla f_i|^h}{|1+\nabla f_i^2|^h}\right).\nabla f_i$$

3.3.1 Vanilla + Momentum

We add a first order momentum term to accelerate our descent and to add a fraction of the previous update to the current update vector. It is henceforth referred to as "my optimizer + momentum".

$$\Delta\theta_{i,t} = -\max\left(\epsilon,\ \frac{|\nabla f_i|^h}{|1+\nabla f_i^2|^h}\right).\nabla f_i - \gamma\ \Delta\theta_{i,t-1}$$

where $\gamma\ \Delta\theta_{i,t-1}$ becomes the first order momentum term to accumulate previous gradients.

3.3.2 Vanilla + EMA

We use the Exponential Moving Average (EMA) of the previous gradients to account for those values in a similar fashion as of RMSProp. It is henceforth referred to as "my optimizer + EMA".

$$E[\nabla f^2]_t = 0.9E[\nabla f^2]_{t-1} + 0.1\nabla f_t^2$$

$$\Delta\theta_i = -\max\left(\ \epsilon,\ \frac{|\nabla f_i|^h}{|1+\nabla f_i^2|^h}\right).\frac{\eta}{\sqrt[2]{E[\nabla f^2]_t+\epsilon}}.\nabla f_i$$

3.3.3 Vanilla + Momentum + EMA

We add a momentum term to the previous update rule. It is henceforth referred to as "my optimizer + Momentum + EMA".

$$E[\nabla f^2]_t = 0.9E[\nabla f^2]_{t-1} + 0.1\nabla f_t^2$$

$$\Delta\theta_{i,t} = -\max\left(\ \epsilon,\ \frac{|\nabla f_i|^h}{|1+\nabla f_i^2|^h}\right).\frac{\eta}{\sqrt[2]{E[\nabla f^2]_t+\epsilon}}.\nabla f_i - \gamma\ \Delta\theta_{i,t-1}$$

In the next chapters, we analyse the effect of the normalizing factor and the momentum and EMA term both theoretically and emperically.

Chapter 4

Theoretical Analysis

4.1 Convergence Validity Theorem

Theorem 4.1.1. *A function $f : \mathbb{R}^n -> \mathbb{R}$ is convex and differentiable, and that its gradient is Lipschitz continuous with constant $L > 0$, i.e. we have $||\nabla f_x - \nabla f_y|| \leq L||x - y||_2$ for any x, y. Then, if we run gradient descent with update rule as $\Delta\theta_i = \frac{|\nabla f_i|^h}{|1+\nabla f_i^2|^h}.\nabla f_i$, it will always converge provided $h > log_2 L - 1$.*

Proof: This step of the proof considers $x \in \mathbb{R}$ and $f : \mathbb{R} -> \mathbb{R}$ with Lipschitz constant as L_i. The same result is expanded to n dimensions in the end.

As ∇f is Lipshitz continuos, we do a quadratic expansion of f around some value $f(x)$ to obtain:

$$f(y) \leq f(x) + \nabla f(x)^T(y - x) + \frac{1}{2}\nabla^2 f(x)||y - x||_2^2$$

$$f(y) \leq f(x) + \nabla f(x)^T(y - x) + \frac{1}{2}L_i||y - x||_2^2$$

For gradient descent step, $y = x^+ = x - \frac{|\nabla f_x|^h}{|1+\nabla f_x^2|^h}.\nabla f_x$.

$$f(x^+) \leq f(x) + \nabla f_x^T(x^+ - x) + \frac{1}{2}L_i||x^+ - x||_2^2$$

$$f(x^+) \leq f(x) + \nabla f_x^T(x - \frac{|\nabla f_x|^h}{|1+\nabla f_x^2|^h}.\nabla f_x - x) + \frac{1}{2}L_i||x - \frac{|\nabla f_x|^h}{|1+\nabla f_x^2|^h}.\nabla f_x - x||_2^2$$

Let $t = \frac{|\nabla f_x|^h}{|1+\nabla f_x^2|^h}.\nabla f_x$,

$$f(x^+) \leq f(x) - \nabla f(x)^T t \nabla f(x) + \frac{1}{2}L_i||t\nabla f(x)||_2^2$$

$$f(x^+) \leq f(x) - (1 - \frac{1}{2}L_i t)\, t||\nabla f(x)||_2^2 \tag{4.1}$$

Now, $t||\nabla f(x)||_2^2$ will be always $+ve$ as both t and $||\nabla f(x)||_2^2$ are always $+ve$, except when $\nabla f(x)$ is 0.

For the term $(1 - \frac{1}{2}L_i t)$ to be $+ve$:

$$0 < 1 - \frac{1}{2}L_i t$$

$$L_i < \frac{2}{t} \tag{4.2}$$

Now, $t = t(\nabla f_x)$ would be maximum at points where $\frac{d(t(\nabla f_x))}{d\nabla f_x} = 0$ and $\frac{d^2 t(\nabla f_x)}{d\nabla f_x^2} < 0$. By solving these, it can be shown that $t(\nabla f_x)$ would be maximum when $\nabla f_x = 1$ and its maximum value would be $\frac{1}{2^h}$. Hence, $\frac{2}{t}$'s minimum value would be 2^{h+1}.

$\therefore$ If $L_i < 2^{h+1}$ or $h > log_2 L_i - 1$, then $(1 - \frac{1}{2}L_i t)t||\nabla f(x)||_2^2$ will always be $+ve$.

From Equation 4.1, we can now follow that objective function value strictly decreases with each iteration of the gradient descent until it reaches the optimal value $f(x) = f(x^*)$. This result only holds if our chosen $h > log_2 L_i - 1$.

We now expand the result to n - dimensions considering $f : \mathbb{R}^n -> \mathbb{R}$ and $x \in \mathbb{R}^n$ with Lipschitz constant as L.

$$L \geq \max_i L_i$$

$$\therefore h > log_2 L - 1 \;\; or \;\; L < 2^{h+1} \;\; for \;\; convergence. \tag{4.3}$$

4.2 Convergence Rate Theorem

Theorem 4.2.1. *A function $f : \mathbb{R}^n -> \mathbb{R}$ is convex and differentiable, and that its gradient is Lipschitz continuous with constant $L > 0$. Then, if we run gradient descent for k iterations with update rule as $\Delta\theta_i = -\max\left(\epsilon, \frac{|\nabla f_i|^h}{|1+\nabla f_i^2|^h}\right).\nabla f_i$, it will lead to a solution $f^{(k)}$ satisfying*

$$f\left(x^{(k)}\right) - f\left(x^*\right) \leq \frac{\left\|x^{(0)} - x^*\right\|_2^2}{2\epsilon k}$$

provided that $h > log_2 L$.

Proof: We try to bound $f(x^+)$, the loss function value at the next step in terms of $f(x^*)$, the optimal value.

As f is convex:

$$f\left(x^*\right) \geq f(x) + \nabla f_x^T \left(x^* - x\right)$$
$$f(x) \leq f\left(x^*\right) + \nabla f_x^T \left(x - x^*\right)$$

Substituting this in to Equation 4.1, we obtain:

$$f\left(x^+\right) \leq f\left(x^*\right) + \nabla f(x)^T \left(x - x^*\right) - (1 - \frac{1}{2}Lt)t||\nabla f(x)||_2^2$$
$$f\left(x^+\right) - f\left(x^*\right) \leq \nabla f(x)^T \left(x - x^*\right) - (1 - \frac{1}{2}Lt)t||\nabla f(x)||_2^2$$

Taking maximum value of $L = 2^h$ and for $t = \frac{1}{2^h}$,

$$f\left(x^+\right) - f\left(x^*\right) \leq \nabla f(x)^T \left(x - x^*\right) - (1 - \frac{1}{2}2^h\frac{1}{2^h})t||\nabla f(x)||_2^2$$
$$f\left(x^+\right) - f\left(x^*\right) \leq \nabla f(x)^T \left(x - x^*\right) - \frac{t}{2}||\nabla f(x)||_2^2$$
$$f\left(x^+\right) - f\left(x^*\right) \leq \frac{1}{2t}\left(2t\nabla f(x)^T \left(x - x^*\right) - t^2\|\nabla f(x)\|_2^2\right)$$
$$f\left(x^+\right) - f\left(x^*\right) \leq \frac{1}{2t}\left(2t\nabla f(x)^T \left(x - x^*\right) - t^2\|\nabla f(x)\|_2^2 - \|x - x^*\|_2^2 + \|x - x^*\|_2^2\right)$$
$$f\left(x^+\right) - f\left(x^*\right) \leq \frac{1}{2t}\left(\|x - x^*\|_2^2 - \|x - t\nabla f(x) - x^*\|_2^2\right)$$

Now, for gradient descent, $x^+ = x - t\nabla f(x)$,

$$f\left(x^+\right) - f\left(x^*\right) \leq \frac{1}{2t}\left(\|x - x^*\|_2^2 - \left\|x^+ - x^*\right\|_2^2\right) \leq \frac{1}{2t_{\min}}\left(\|x - x^*\|_2^2 - \left\|x^+ - x^*\right\|_2^2\right)$$

$$f\left(x^+\right) - f\left(x^*\right) \leq \frac{1}{2\epsilon}\left(\|x - x^*\|_2^2 - \left\|x^+ - x^*\right\|_2^2\right)$$

This inequality holds for x^+ on every epoch of gradient descent. Summing over multiple epochs, we can deduce:

$$\begin{aligned}\sum_{i=1}^{k} f(x^{(i)}) - f\left(x^*\right) &\leq \sum_{i=1}^{k} \frac{1}{2\epsilon}\left(\left\|x^{(i-1)} - x^*\right\|_2^2 - \left\|x^{(i)} - x^*\right\|_2^2\right) \\ &= \frac{1}{2\epsilon}\left(\left\|x^{(0)} - x^*\right\|_2^2 - \left\|x^{(k)} - x^*\right\|_2^2\right) \\ &\leq \frac{1}{2\epsilon}\left(\left\|x^{(0)} - x^*\right\|_2^2\right)\end{aligned}$$

Using the fact that f is decreasing on every iteration, we can conclude that,

$$\begin{aligned}f\left(x^{(k)}\right) - f\left(x^*\right) &\leq \frac{1}{k}\sum_{i=1}^{k} f\left(x^{(i)}\right) - f\left(x^*\right) \\ &\leq \frac{\left\|x^{(0)} - x^*\right\|_2^2}{2\epsilon k}\end{aligned}$$

Chapter 5

Empirical Analysis

We evaluate the performance of the optimizer first on a 2-dimensional set of non-convex functions which are traditionally considered difficult to optimize. Then we evaluate the performance on both shallow and deep neural networks for several classification and regression tasks along with training word embeddings using unsupervised methods.

5.1 Empirical Analysis over $2D$ non-convex functions

5.1.1 Test functions set

We evaluate the performance on 30 $2D$ non-convex functions which have many local minima, are bowl-shaped, are plate-shaped, are valley-shaped, have steep ridges/drops and are considered difficult to optimize by gradient descent based methods. They are sourced from Virtual Library of Simulation Experiments: Test Functions and Datasets [SB]. Two such functions are shown below:

1. Easom function:

$$f(x, y) = -\cos(x)\cos(y)\exp^{-(x-\pi)^2-(y-\pi)^2}$$

   The Easom function has several local minima. It is unimodal, and the global minimum has a small area relative to the search space. It has global minima $f(x^*) = -1$ at

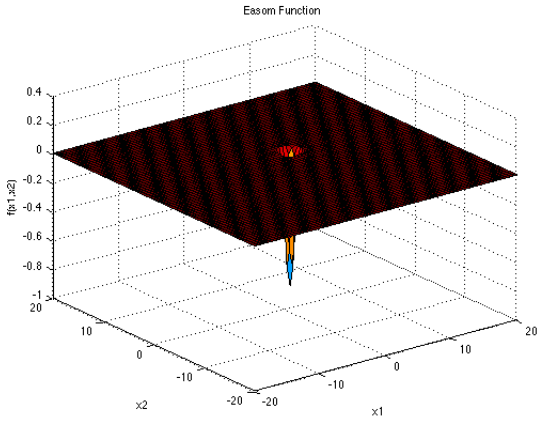


Fig. 5.1 Easom function

$x^* = (\pi, \pi)$.

2. Bohachevsky function:

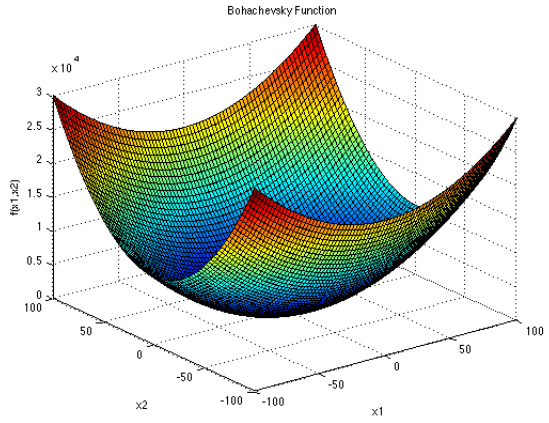


Fig. 5.2 Bohachevsky function

$$f(x, y) = x^2 + 2y^2 - 0.3\cos(3\pi x) - 0.4\cos(4\pi y) + 0.7$$

The Bohachevsky functions all have the same similar bowl shape. It has global minima $f(x^*) = 0$ at $x^* = (0, 0)$.

The rest of the functions can be view from Virtual Library of Simulation Experiments:

Test Functions and Datasets [SB][1].

5.1.2 Validation parameters

We define three empirical validation parameters to score our optimizer against standard optimizers. They are listed below:

1. Does the descent converge?

   Descent converge is considered to be true if there exists some $c \in \mathbb{N}$ such that $|x_{t+1} - x_t| \leq k \ \ \forall \ \ t > c$, where k is the allowed error and then c becomes the number of epochs required to converge.

2. Score over number of epochs required to converge

$$s_1 = \begin{cases} \frac{1}{c} & \text{if } \exists \ \ c : |x_{t+1} - x_t| \leq k \ \ \forall \ \ t > c \ \ \text{(descent converges)} \\ 0 & \text{otherwise (descent does not converge)} \end{cases}$$

3. Score over converging at global minimum?

$$s_2 = \begin{cases} 10 & \text{if } \ |x_c - x^*| \leq k^{'} \\ 0.1 & \text{otherwise} \end{cases}$$

   where $k^{'}$ is the allowed error, x^* is the global minima and c is the number of epochs required to converge.

The score of the optimizer over single test function with one particular initialization point in $\mathbb{R}^2$ can be calculated as $s_1 s_2$. Overall score is calculated as average over all the test functions starting from multiple initialization points.

$$\text{Score} = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} s_1(f_i, x_{i,j}) s_2(f_i, x_{i,j})}{n + m} \tag{5.1}$$

[1] `https://www.sfu.ca/~ssurjano/optimization.html`

where, f_i is the i^{th} test function and $x_{i,j}$ is the j^{th} initialization point for the i^{th} test function. During testing, $x_{i,j}$ are randomly initialized $\forall\ \ i, j$.

5.1.3 Sample Descent graphs for Easom function

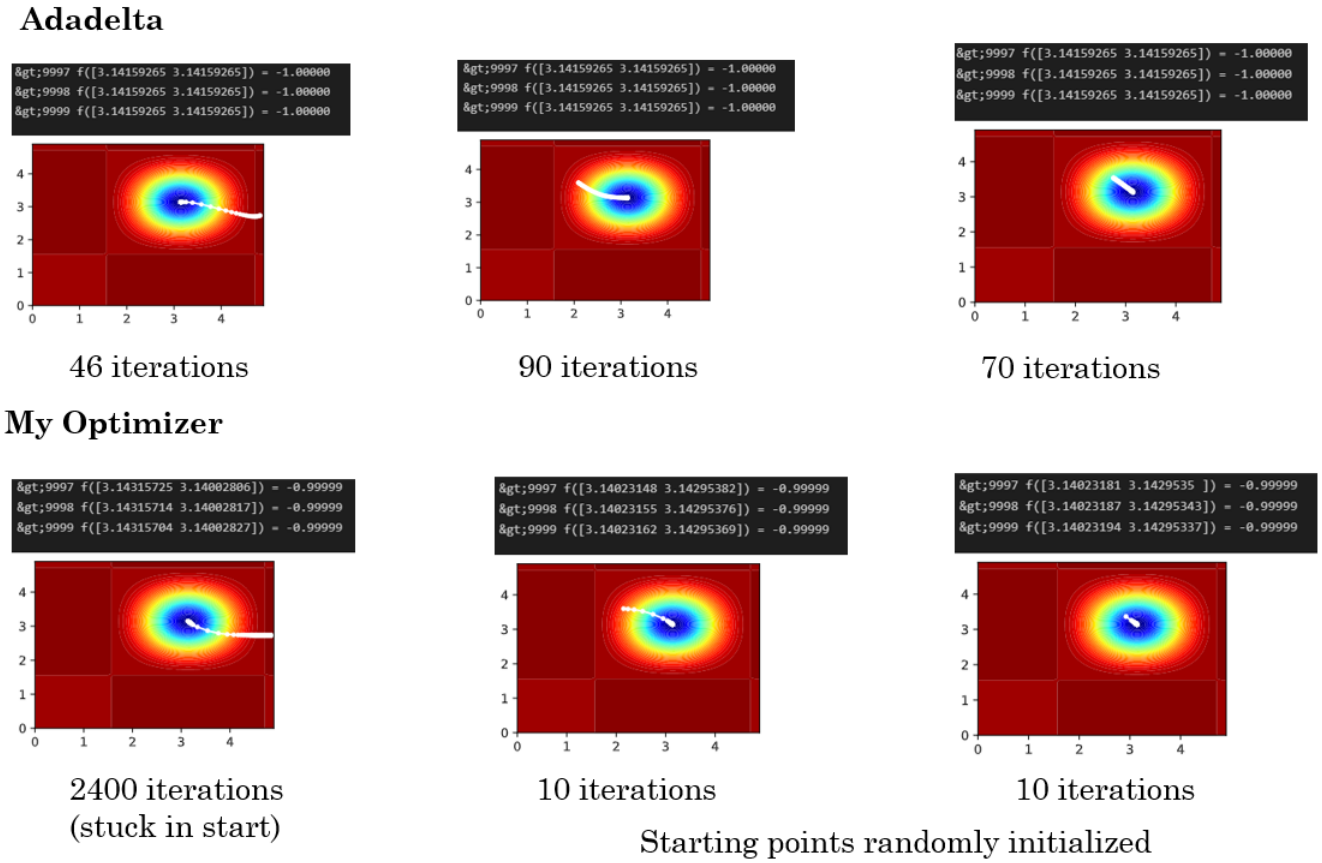


Fig. 5.3 Descent graph for Easom function

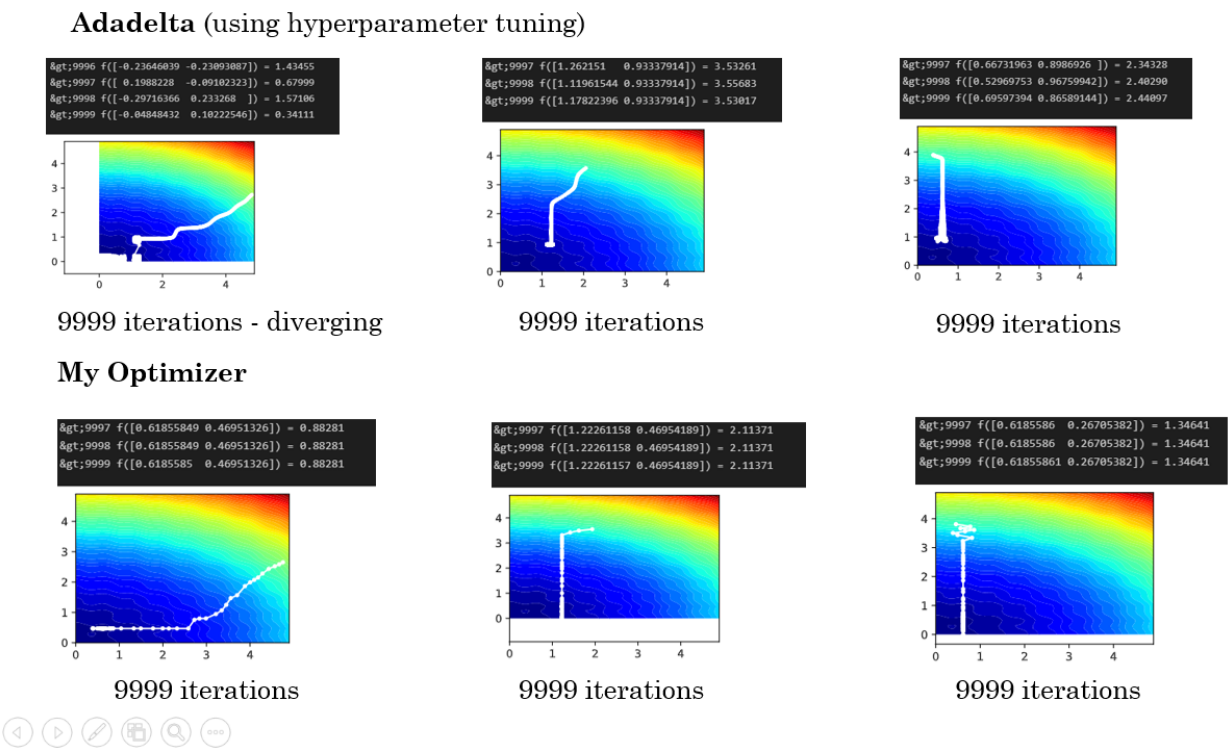


Fig. 5.4 Descent graphs for Bohachevsky function

5.1.4 Sample Descent graphs for Bohachevsky function

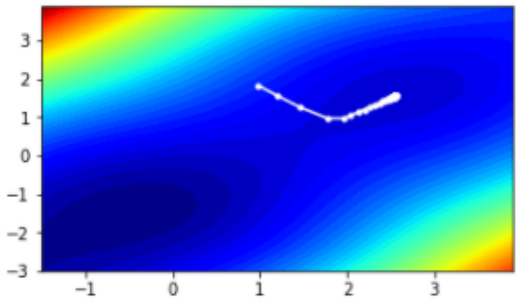
5.1.5 Sample descent for other tests showing the convergence and score parameters

5.1.6 Overall Results

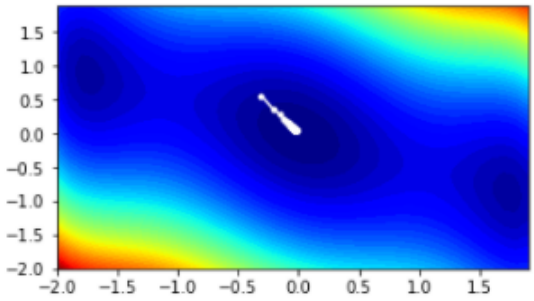
The average score over all functions and initialization points as shown in Equation 5.1 is calculated and presented in the below table. The code to run the descent and re-evaluate this table is available here [2].

[2]https://github.com/fliptrail/btp

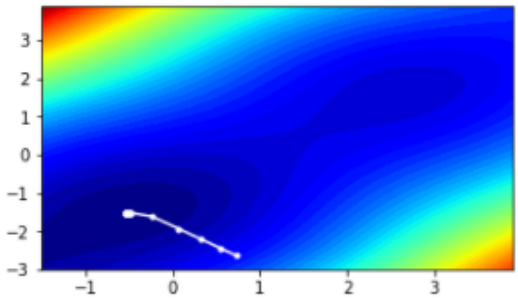
Using function: MCCORMICK with seed: 1
Converged: True
Converged at Global Minima: False
Number of epochs required to converge: 6
Score: 0.03333333333333333



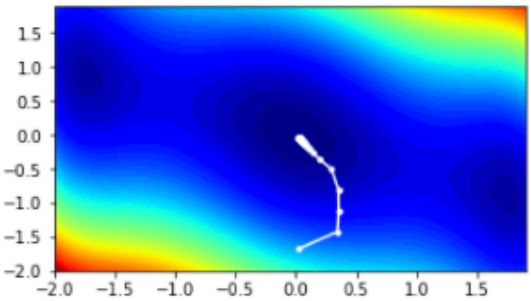
Using function: THREEHUMP_CAMEL with seed: 1
Converged: True
Converged at Global Minima: True
Number of epochs required to converge: 3
Score: 3.3333333333333335



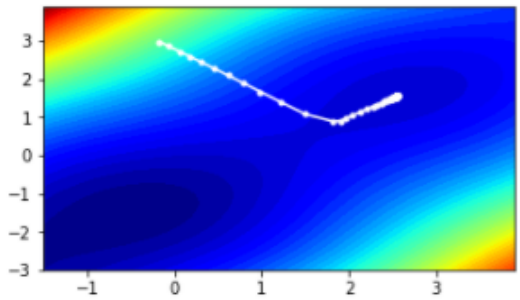
Using function: MCCORMICK with seed: 2
Converged: True
Converged at Global Minima: True
Number of epochs required to converge: 7
Score: 1.4285714285714286



Using function: THREEHUMP_CAMEL with seed: 2
Converged: True
Converged at Global Minima: True
Number of epochs required to converge: 7
Score: 1.4285714285714286



Using function: MCCORMICK with seed: 5
Converged: True
Converged at Global Minima: False
Number of epochs required to converge: 13
Score: 0.015384615384615385



Using function: SPHERE with seed: 5
Converged: True
Converged at Global Minima: True
Number of epochs required to converge: 19
Score: 0.5263157894736842

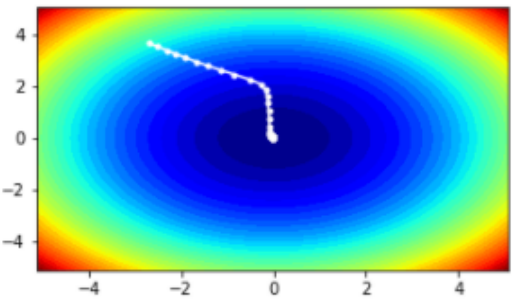


Fig. 5.5 Sample descent for other tests showing the convergence and score parameters

Optimizer	Average Score	Number of divergences
SGD	0.314	28
SGD with Momentum	0.322	41
Adagrad	0.282	14
RMSProp	0.520	8
Adam	0.885	4
My optimizer	0.825	**0**
My optimizer + Momentum	0.641	8
My optimizer + Momentum + EMA	**0.944**	**0**

Table 5.1 Score of various optimizers after hyperparameter tuning over 2D non-convex test suite. The average is taken over $30 * 10 = 300$ runs.

5.2 Empirical Analysis over Neural Networks

We evaluate the performance of our optimizer over a suite of shallow, deep and specialized neural networks. The networks and the datasets used are highlighted below.

5.2.1 Datasets used

The datasets used for the tasks are highlighted below:

1. Airfoil Self-Noise Data Set [DG17]

   It is a NASA dataset obtained from a series of aerodynamic and acoustic tests of two and three-dimensional airfoil blade sections conducted in an anechoic wind tunnel. It comprises of different size NACA 0012 airfoils (n0012-il) at various wind tunnel speeds and angles of attack. The span of the airfoil and the observer position were the same in all of the experiments. [Fed]

2. Fashion-MNIST [XRV17]

   Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. The authors intend Fashion-MNIST to serve as a direct drop-in replacement for the original MNIST dataset

for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

3. The CIFAR-10 [Kri09]

   The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. It was collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

4. The Penn Treebank [MSM93]

   The Penn Treebank (PTB) project selected 2,499 stories from a three year Wall Street Journal (WSJ) collection of 98,732 stories for syntactic annotation. The Penn Treebank, in its eight years of operation (1989–1996), produced approximately 7 million words of part-of-speech tagged text, 3 million words of skeletally parsed text, over 2 million words of text parsed for predicateargument structure, and 1.6 million words of transcribed spoken text annotated for speech disfluencies. The material annotated includes such wide-ranging genres as IBM computer manuals, nursing notes, Wall Street Journal articles, and transcribed telephone conversations, among others.

5.2.2 Empirical analysis over specialized neural networks

We run various optimizers including ours and its variations on logistic regression, linear regression, shallow regression, deep neural regression, single hidden layer classification, deep classification and unsupervised word embeddings training using Continuos-Bag-Of-Words (CBoW) [MCCD13] method. The regression tasks are run over Airfoil dataset. The classification tasks are run over Fashion-MNIST and CIFAR-10 datasets. The word embeddings training uses The Penn Treebank. Input features in Airfoil, Fashion-MNIST and CIFAR-10 are both unnormalized and mean and standard deviation normalized. These

datasets are highlighted in Section 5.2.1. The results showing the convergence epochs are in Table 5.2 and the convergence graphs are shown in the figures below.

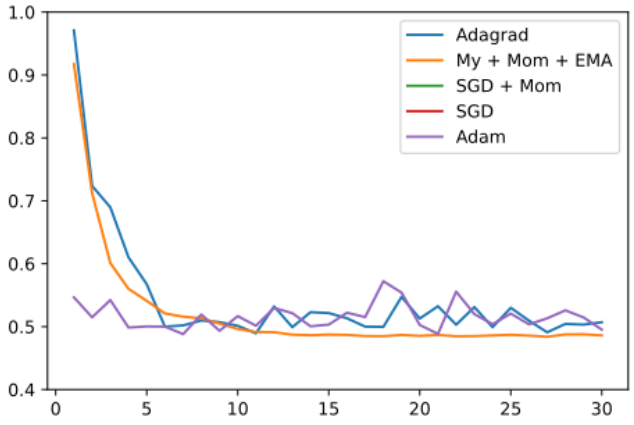


Fig. 5.6 Convergence graph for shallow regression over Airfoil (Normalized)

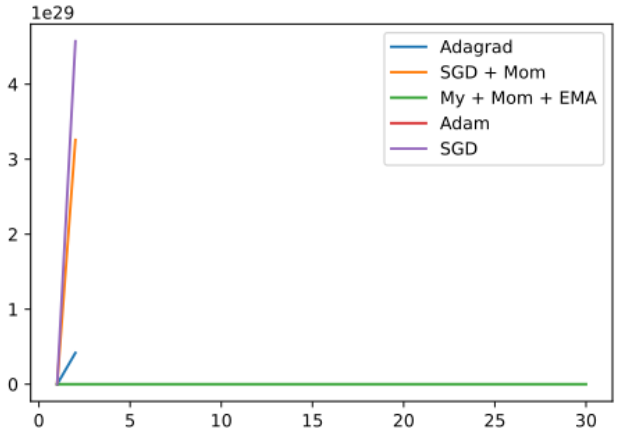


Fig. 5.7 Convergence graph for shallow regression over Airfoil (Unormalized)

5.2.3 Overall Results

-	SGD	Adagrad	Adam	My optimizer	My opt. + Mom. + EMA
Shallow Regression over Airfoil	inf	1	1	1	1
Shallow Regression over Airfoil (Normalized)	2	1	1	1	1
Deep Regression over Airfoil	inf	2	2	2	1
Deep Regression over Airfoil (Normalized)	6	5	1	5	3
Shallow Classification over Fashion-MNIST	inf	inf	inf	3	3
Shallow Classification over Fashion-MNIST (Normalized)	6	2	2	3	3
Deep Classification over Fashion-MNIST	inf	23	inf	40	14
Deep Classification over Fashion-MNIST (Normalized)	4	3	3	5	3
Shallow Classification over CIFAR-10 (Normalized)	32	18	20	52	19
Deep Classification over CIFAR-10 (Normalized)	44	40	18	28	24
Word Embeddings using Penn Treebank (CBoW)	inf	122	90	151	111

Table 5.2 Convergence epochs for optimizers over Neural Networks

Chapter 6

Training Framework API

We create a general neural network training framework in Python over Tensorflow and Keras such that given any neural network, any dataset, any loss function and any optimizer; it should be able to train the parameters of the network and plot the loss curve.

The framework is available here for download. [1]

6.1 Implemented APIs

6.1.1 `get_classification_data(batch_size, normalized)`

Arguments:

1. `batch_size (int)`: Specifies the batch size for the returned data iterators
2. `normalized (bool)`: Specifies if the input features should be mean and standard deviation normalized

Returns:

1. Training data iterator (`tf.data.Iterator`): Tensorflow data iterator to iterate over `tf.data.Dataset` for training data.

[1] https://github.com/fliptrail/btp

2. Test data iterator (`tf.data.Iterator`): Tensorflow data iterator to iterate over `tf.data.Dataset` for test data.

6.1.2 `get_regression_data(batch_size, normalized)`

Arguments:

1. `batch_size (int)`: Specifies the batch size for the returned data iterators
2. `normalized (bool)`: Specifies if the input features should be mean and standard deviation normalized

Returns:

1. Training data iterator (`tf.data.Iterator`): Tensorflow data iterator to iterate over `tf.data.Dataset` for training data.
2. Test data iterator (`tf.data.Iterator`): Tensorflow data iterator to iterate over `tf.data.Dataset` for test data.

6.1.3 `general_trainer(model, data_iter, loss, optimizer, hyperparameters, epochs)`

Arguments:

1. `model (tf.keras.Model)`: The neural network in `tf.keras.Model` format either built with the Functional API or the Sequential model (`tf.keras.Sequential()`). It can accept any neural network in this format.
2. `data_iter`: Data iterator of `tf.data.Iterator` format.
3. `loss (tf.keras.losses)`: The loss function to use to train the model. Example: `tf.keras.losses.BinaryCrossentropy()`

4. `optimizer`: A custom or in-built class to specify the update rules. The class should accept hyperparameters as a `dict`.

5. `hyperparameters (dict)`: A dictionary containing all the required parameters for the optimizer.

6. `epochs (int)`: Number of epochs to train upto.

6.1.4 Example Usage

```
data_iter, test_iter = get_classification_data(batch_size=256, normalized = True)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
general_trainer(fashion_mnist, data_iter, loss, my_optimizer, {'power_factor': 2}, 40)
```

Fig. 6.1 Example showing usage of the Training Framework

6.2 Loss function plot

While training the architecture, a curve of loss function vs number of epochs is generated as shown in the Figure 6.2.

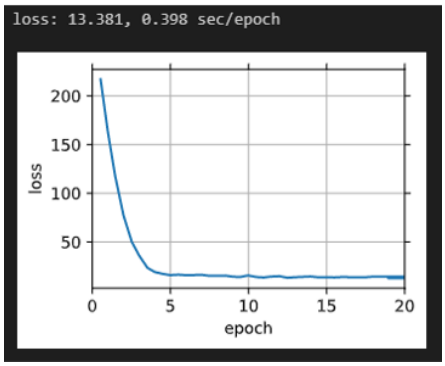


Fig. 6.2 An example of the loss function curve generated from the training framework.

Chapter 7

Conclusion and Future Work

The thesis presents a novel normalizing factor for the gradient descent update rule. By combining it with other gradient descent optimizers, we are able to get better update rules which aim to reduce divergence. We analyse the effects of this normalization factor both theoretically and empirically. Theoretically, we prove its convergence and converge bounds and empirically we test it over $2D$ non-convex functions and on neural network based regression, classification and word embeddings training problems. We also provide a Training Framework to train Tensorflow and Keras built models using our optimizer.

Future scope of this work includes:

1. Proving the REGRET bounds of this optimizer
2. Proving the series-limit bounds of this optimizer
3. Using the optimizer over Attention-based models
4. Building a better benchmarking strategy to compare performance over neural networks

References

[BN12] Sébastien Bubeck and Cesa-Bianchi Nicolò. 2012.

[DG17] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[DHS11] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.

[Fed] Fedesoriano. Airfoil dataset kaggle.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[Kri09] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[MSM93] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[Qia99] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999.

[Rud16] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016.

[SB] S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets. Retrieved May 2, 2022, from `http://www.sfu.ca/~ssurjano`.

[XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[Zei12] Matthew D. Zeiler. Adadelta: An adaptive learning rate method, 2012.