Algorithmic Behaviours of Adagrad in Underdetermined Linear Regression

by

Andrew Rambidis

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Data Science

Waterloo, Ontario, Canada, 2023

Abstract

With the high use of over-parameterized data in deep learning, the choice of optimizer in training plays a big role in a model's ability to generalize well due to the existence of solution selection bias. We consider the popular adaptive gradient method: Adagrad, and aim to study its convergence and algorithmic biases in the underdetermined linear regression regime. First we prove that Adagrad converges in this problem regime. Subsequently, we empirically find that when using sufficiently small step sizes, Adagrad promotes diffuse solutions, in the sense of uniformity among the coordinates of the solution. Additionally, when compared to gradient descent, we see empirically and show theoretically that Adagrad's solution, under the same conditions, exhibits greater diffusion compared to the solution obtained through gradient descent. This behaviour is unexpected as conventional data science encourages the utilization of optimizers that attain sparser solutions. This preference arises due to some inherent advantages such as helping to prevent overfitting, and reducing the dimensionality of the data. However, we show that in the application of interpolation, diffuse solutions yield beneficial results when compared to solutions with localization; Namely, we experimentally observe the success of diffuse solutions when interpolating a line via the weighted sum of spike-like functions. The thesis concludes with some suggestions to possible extensions of the content in future work.

Acknowledgements

First and foremost, none of my accomplishments nor achievements would have been possible if it was not for the amazing support provided by my parents and sister. Mom, Dad, I am thankful for all the guidance you've given me, and I am very grateful that you've given me a healthy and stable environment which allowed me to grow and pursue the things that bring me joy. To my sister Emily, thank you for always having my back! It brings me so much joy that I have a sister in which I can trust to be by my side in both the ups and downs that life has to offer.

To my significant other, Jiayi; Thank you for being in my life and always pushing me to be the best version of myself. I am grateful that you accept me for who I am, and appreciate all the times you've pushed me out of my comfort zone in order to grow as a person. I am thankful to have you in my life, and I look forward to the journeys to come.

It is without question that I am truly thankful to my best friend Matthew Flood for providing me a space where I can be me without having to worry about the baggage tied to societal conformity. Any thing I do with you is a great time, and I know that, even when I'm going through tough times, your door is always open for me. Thank you for the countless hours of laughter, joy, and fun.

To the rest of my close Montreal friend gang, Conrad, Isi, Liam, Kaitlin, Marco, Dave, and Andrew, it is always a pleasure to hang out and have fun, whether that be our classic game nights, restaurants, or cabin trips. Thank you for being an avenue of joy and fun separate from my academic work. This diversity of joys in life is what prevents me from burning out and enjoying life.

I'd like to extend a huge thank you to my Master's supervisor Stephen Vavasis. Steve, I truly feel lucky to have worked under you. I've learned so much from what seems like your endless, diverse and vast source of knowledge. You have given me a strong base towards my goal of becoming a problem solver. While I still have so much to learn and grow, your guidance has shaped my character to be more rigorous, thorough, and critical in the work I produce. I'd also like to point out that your guidance has aided me in the skill of asking the right questions. This was a skill I really wanted to improve on and you have steered me towards the right path. If it was not obvious enough already, you inspire me to become a bright mind and mathematician, and are at a level I strive to reach and possibly surpass. Your diverse knowledge is a strong indicator for me that I too can be a strong mind in many fields. Thanks Steve!

To my thesis readers Walaa and Yaoliang, both of you have had great impact on my path to become a better problem solver. On top of the efforts you guys have put into ensuring my thesis is of adequate quality, I want to express that the classes I took with you have amplified my excitement and joy in the field of optimization. To Yaoliang, while we never had the opportunity to have an in-person course together due to Covid-19, your CS 794 course was taught amazingly, and just from your lecture recordings alone you have inspired me to be as knowledgeable as you are in whatever field I end up calling my own. I strive to become someone who is knowledgeable enough such that they can be coherent and clear when talking about a subject as you did in your lectures. To Walaa, thank you for all of your patience and guidance throughout CO 769. The content was challenging but extremely interesting and your teaching style definitely played a huge role into why that was. I still recall attending every office hour like a ritual so that I can get the opportunity to pick your brain and learn more. As with Steve, and Yaoliang, you are a role model in what I work hard to become.

To my office mates and friends, Matt, Tyler, Tina, Kartik, Leo, and Martin, thank you all for being by my side throughout this journey. Life would not be as enjoyable if not for you guys. I loved chatting with you during lunches, hanging out at restaurants and bars at the end of a good week of work, and most of all feeling included in something social. As a data science student, you have made my integration with the C&O department very easy to the point where I sometimes forget I am doing a MMath DS and not a MMath CO!

Finally, I want to thank the the Data Science and C&O staff which enabled me to have a seamless experience with all things technical and administrative pertaining to my degree.

Dedication

This is dedicated to my growth of becoming a better critical thinker and problem solver.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The use of machine learning techniques have exploded in the past decade, and many industries are attempting to leverage the power present in these techniques today [41]. A certain subset of these techniques known as *deep learning* (DL) has taken the world by storm due to their impressive feats in computer vision [14] and natural language processing [54, 45], to name a few. The potential capabilities achievable in this regime of *artificial intelligence* (AI) has sparked a massive wave of interest which, in turn, has sparked a revolution in machine learning research and development. Machine learning is a category of AI that allows a machine to learn patterns hidden in data to be leveraged in completing a task in an automated fashion. Deep learning is a subset of machine learning in the sense that its techniques for pattern recognition are built from a class of model known as a neural network. The *deep* in deep learning comes from the fact that modern neural networks are trained with a large amount of parameters and layers and are very complex in nature.

To achieve the generalization performance seen in modern neural networks, the machine needs to choose its parameters in a way that its output yields the best possible results. The best possible result depends on the task at hand, for example, if we are classifying whether an image is that of a dog or cat, then the best possible result would be the machine properly classifying the images. The idea of finding the best parameters can be framed as an optimization problem where the machine is required to pick the parameters that minimize the amount of errors made. This is done by minimizing something called the *loss function*. The loss function tells the machine how well it is performing based on the selected parameters. The act of minimizing this loss function is called *training* the model. The way one trains such a model is by using something called an *optimizer* whose goal is to find the parameters that minimize the loss function. The problem is, many state of the art deep learning models, such as OpenAI's GPT-3 model [10], contain billions of parameters

[55, 10]. In comparison to the amount of data available to be used to train such models, we are left with having to estimate more parameters than there is data to estimate from. This causes the space of solutions (i.e., space of optimal parameters) to be very large. Due to this, different optimizers will tend to bias towards a particular choice of solution which one calls the *implicit bias* of the optimizer. Due to these biases, the choice of optimizer plays a key role in the generalization performance of a model.

A class of optimizers that have shown to enable a model to have strong generalization performance are gradient-based optimizers, in particular, *stochastic gradient descent* (SGD) and its variants. Apart from their ability to choose solutions that generalize well, SGD and its variants are also favourable due to their ability to be efficient in large-scale problems [38]. The question becomes, given we know that an optimizer will have such biases, can we characterize these behaviours? That is, what special minimum will it choose from given the vast amount of choices, and does this choice enable strong generalization abilities?

SGD has been a central figure in the study of implicit biases in the regime of neural networks due to its ability to pick "favourable" parameters that enables a model to generalize well [34]. Because of this, there is a rich theory on its implicit biases all while new theoretical and experimental results continue to be exposed. However, a new stream of SGD variants known as adaptive-gradient methods, such as Adagrad [18] and Adam [36], have taken the charge as the de facto optimizers used in current neural network training. Yet, there is a lack of theoretical analysis on the generalization behaviours of these adaptive algorithms that warrants the same treatment as SGD.

The motivation behind this thesis comes from the desire to study and understand adaptive gradient methods, and to perform theoretical and experimental analyses on their convergence behaviours in an overparameterized regime like that of neural networks in an attempt to discover their generalization behaviours.

1.1 Thesis Outline

This thesis aims to study algorithmic behaviours of adaptive gradient optimization algorithms, specifically with a focus on the Adagrad algorithm. Prior to delving into the main content, Chapter 2 briefly covers related works that helped inspire the work done in this thesis, and to catalogue such works with the intention of informing the reader of the existence of such works which are connected to this area of research. We officially begin in Chapter 3 by covering the basics we'll need to grasp the contents of later chapters. Section 2.1 covers a primer on supervised deep learning with an emphasis on the methods and difficulties of training them. Section 3.2 covers the two fundamental optimization algorithms: gradient descent (section 3.2.1), and stochastic gradient descent (section 3.2.2), in which the construction of the adaptive methods introduced in Chapter 4 are based on. Section 3.2.4 briefly discusses subgradients and the subgradient descent algorithm. Sections 3.3, 3.4 cover the basics of online learning used in Chapter 4, and radial basis function interpolation for which a variant is used in our experiments in section 6.2, respectively. In Chapter 4, we introduce four highly used adaptive methods starting from the first adaptive variant known as AdaGrad [18], all the way to the current trending optimizer in DL known as Adam [36]. Moving forward, we set our focus on exploring the algorithmic behaviours of Adagrad in the underdetermined linear regression regime. Chapter 5 goes over our theoretical results which mainly include a proof of convergence for Adagrad, and a proof related to our main experimental finding seen in Chapter 6 section 6.1 which shows that Adagrad promotes diffuse solutions in the underdetermined linear regression regime. Section 6.2 shows an application where diffuse solutions tend to interpolate a line better using a weighted sum of spike-like functions. Chapter 7 wraps up the thesis and presents possible avenues of future work.

Chapter 2

Related Works

Connection Between Neural Networks and Linear Models

The overarching theme of this research is the study of interpolators, specifically complex deep neural networks, which have been seen to generalize well in practice contrary to conventional statistical beliefs. These deep NNs tend to be hard to analyze theoretically so instead, the studies usually limit themselves to the least-squares regime. This choice of regime is not studied arbitrarily, and there exists work connecting the linear regime to the NN regime. The work by Jacot et al. [31] aids in showing this connection by studying the behaviours of infinite-width neural networks. They showed that for an infinite-width feed-forward neural network with certain activation functions, the dynamics of the network can be approximated using a kernel method. Dubbed the Neural Tangent Kernel (NTK), this kernel allows one to describe the behaviour of such a network as a Gaussian process (GP) where the NTK captures the covariance of said GP. The NTK can be thought of as a kernel matrix (seen in kernel methods), and therefore one can use linear models to learn the non-linear patterns of the data by applying these models on the function space induced by the kernel function.

Understanding the Behaviours of Deep Neural Network Interpolators

Following the theme of understanding the "why" in strong generalization performance of interpolators in the NN regime, Hastie, et al. [27] studies the behaviour of NN interpolators via the ℓ_2-norm least squares regression regime trained using gradient descent. In the paper, the authors study the prediction risk on the out-of-sample (test) data in the asymptotic regime for both the overparameterized and underparameterized regime. They conduct this study using data generated from two types of models: the first is a linear model where the samples are of the form $\mathbf{x}_i = \Sigma^{1/2}\mathbf{z}_i$, where the $\mathbf{z}_i \in \mathbb{R}^d$ such that the coordinates are i.i.d and generated using a random normal with mean zero and unit variance, and $\Sigma \in \mathbb{R}^{d\times d}$ is deterministic and positive semi-definite; the second is a non-linear model of the form $\mathbf{x}_i = \sigma(W\mathbf{z}_i)$. In the non-linear model, z_i is constructed the same as the linear model, σ is an activation function, and $W \in \mathbb{R}^{p\times d}$ has entries generated from a random normal with mean zero and variance $(1/d)$. For brevity, we focus on the result of the linear data.

In their asymptotic setup, the number of observations, n, and the number of features, d, diverge $(n, d \to \infty)$, such that the ratio between them converges to a constant: $d/n \to \gamma > 0$. When $\gamma < 1$, one is working in the underparameterized regime. When $\gamma > 1$, then one is working in the overparameterized regime. From this setup, the following results were found for the linear modelled data: In the underparameterized regime $(\gamma < 1)$, as $n, d \to \infty$ such that $d/n \to \gamma < 1$ then the risk offered by the minimum ℓ_2-norm solution of least squares, $\hat{\theta}$, depends purely on variance and not bias. Likewise, in the overparameterized regime $(\gamma > 1)$ the risk offered from $\hat{\theta}$ depends on both bias and variance.

The authors limit their study to the minimum ℓ_2-normed solution (i.e. gradient descent) and do not consider the effects of other first order methods such as the adaptive ones we wish to study. It is worth mentioning that similar work was produced by Advani and Saxe [1] which also focused on the asymptotic analysis but in a broader setting.

Another behavioural aspect of the out-of-sample prediction risk which garners interest is the study of the bias-variance trade off and the rise of the double-descent phenomenon exhibited by NN in the overparameterized regime. This phenomenon was first posited by Belkin et al. [4] where they provided evidence of the existence of the double descent curve for a wide range of ML models and data sets. Classically, the bias-variance trade off is a summary term for the idea that one must find a balance between the bias and variance of their model in order to avoid both under- and over-fitting the training data. If one were to plot the test error based on the capacity of the model (the larger the capacity, the lower the bias, and the higher the variance) one should expect to see a U-shaped graph indicating that low capacity models which underfit the training data perform poorly, large capacity models which overfit the training data perform poorly, and existence of a sweet spot between bias and variance exists in which we gain the best out-of-sample prediction performance. The double-descent phenomenon arises in the over-parameterized regime where the out-of-sample prediction error exhibits a second descent after the original U-shape. In other words, in the right setting, and for certain classes of models, having extremely high capacity will allow the emergence of a second round of descent. Explicitly, what was seen is that as the number of features d approaches the number of observations n from below, we experience the typical U-curve in the test prediction risk plot. When $d = n$ the risk diverges, and as d continues to grow past n $(d > n)$, the test prediction risk begins to descend again. This reconciles the strong generalization performance witnessed in interpolators and the bias-variance trade off.

From this discovery, Belkin et al. [5] theoretically show the double-descent phenomenon with respect to the ℓ_2-norm least-squares regime studied by Hastie et al. [27]. The authors, however, focuses on finite-sample analysis where the features and and labels are jointly Gaussian. Specifically, they prove that, under this setting, around when $d = n$, the out-of-sample prediction risk is infinite, and when $d > n$ the risk decreases (i.e. achieve the double-descent). Additionally, when the signal to noise ratio (SNR)[1] is high, the authors prove that the minimum risk is achieved in the second descent regime where $d > n$. This result pertaining to SNR was also witnessed in Hastie et al. [27] under their linearly modelled data regime mentioned above.

Implicit Bias and Convergence Behaviour

So far, the works mentioned above used gradient descent as a means of optimizer when analyzing these regimes. Ideally, we'd also like to study these regimes under different first-order methods, specifically the adaptive gradient methods. The PhD dissertation of Vastal Shah [52, Chapter 3] contains a chapter on the convergence behaviour of adaptive gradient methods in the overparameterized linear regression regime. Specifically, the author aims to compare the convergence behaviour of different classes of adaptive methods and how they compare to the non-adaptive methods. Denoting D_k to be a general preconditioner matrix, Shah shows the existence between two classes of preconditioner matrices: the first class behaving similar to (stochastic) gradient descent (GD) in that they converge to the minimum ℓ_2-norm solution, and the second are those that do not.

For the class of D_k such that the convergence behaviour is not the same as the non-adaptive methods, the difference comes from the fact that, for such a class of preconditioner, the solution will have an in-span component and an out-of-span component. Here, by "span" we are referring to the span of the data matrix. The author presents a closed form solution of an iterate, $\mathbf{w}(T)$ at some time T and shows that for correct choice of D_k, the iterates $\mathbf{w}(t)$ can lie outside the span of the data.

Shah also considers the overparameterized regularized linear regression regime. Under this regime, the author shows that if the preconditioner matrix D_k is positive definite, the adaptive method will converge to the same solution as that of (stochastic) GD (under the same regularized regime).

Wang et al. [56] also analyze the implicit bias of adaptive methods, however, they focus their study on homogeneous neural networks[2] with separable data under the logistic

[1]SNR is the ratio between relevant information compared to the irrelevant information of the data.

[2]Homogeneous neural networks are those in which all the layers (outside of the input) share the same number of neurons and activation functions.

regression setting. Similar to Shah [52, Chapter 3], the authors aim to study two classes of preconditioners: the first class are those from the Adagrad [18] family in which the preconditioner utilises the average of past squared gradients, and those that utilise exponentially weighted moving averages of the gradients such as RMSProp [29] and Adam [36] (see chapter 4 for more information on these algorithms). In the non-adaptive case, it is well known that gradient descent converges to the max-margin solution[3] under logistic regression for separable data [53], and furthermore, was shown to maximize the margin of homogeneous neural networks with separable data under logistic regression [40].

In the logistic setting for separable data, the global minimizer of the logistic loss function is not attainable, and, therefore, the convergence behaviours of algorithms are described via the asymptotic direction of their iterates: $\lim_{k\to\infty} \frac{\mathbf{w}_k}{\|\mathbf{w}_k\|_2}$. Under this setting, the authors show that the asymptotic direction of Adagrad is affected by its preconditioner, while the direction given by RMSProp is independent of its preconditioner, and, in fact, follows the same direction as gradient descent and thus yields the max-margin solution. The max-margin solution is known to generalize well which shows that, under this regime, RMSProp tends to generalizes better than Adagrad.

On the topic of the Adagrad family of adaptive methods, Antonakopoulos et al. [2] aimed at studying the tradjectory of Adagrad and whether they avoid saddle points. They focus on the general unconstrained non-convex problem in $\mathbb{R}^d$, and show that the trajectories induced by the family of Adagrad algorithms avoid saddle points from almost any initial condition. To be able to perform proper analysis of said result, the authors showed the existence of a positive definite limit for the preconditioner matrix which enabled the ability to perform stability analysis[4] on the preconditioner. From this analysis, the authors were able to produce a stable manifold theorem[5] with respect to Adagrad which was then additionally extended globally. This allowed them to reach their conclusion.

Finally, we add here that a recent work by Défossez, et al. [20] provides proof of convergence for Adagrad and Adam when applied to smooth (possibly non-convex) objective functions with bounded gradient. We go into more detail about their result pertaining to Adagrad at the end of Section 5.3.

[3]The max-margin solution refers to the solution that emits a decision boundary that maximizes the margin between classes of features.

[4]Stability analysis refers to analyzing the behaviour of a system and how likely it is to converge to a desirable state in the presence of disturbances. With respect to saddle points, stability analysis of a trajectory can provide insight on the trajectory's behaviour near the saddle point.

[5]This stable manifold theorem provides existence of a stable manifold for Adagrad and therefore must imply that the set of initial values which lead to a saddle point is of measure 0.

Generalization Performance of Adaptive vs. Non-Adaptive Methods

While it has been seen that the convergence speeds for adaptive methods such as Adagrad, RMSProp, and Adam, tend to outperform the non-adaptive methods in the deep learning setting, Wilson et al. [57] argue that this may not be the case when it comes to generalization performance. The authors construct an empirical example [57, Section 3.3] of a binary classification where the data is linearly separable where Adagrad, RMSProp, and Adam each fail to classify out-of-sample data with probability close to 0.5; Meanwhile SGD attains zero out-of-sample prediction error. This holds even when the training losses for the aforementioned adaptive methods are lower than that of SGD. Furthermore, the authors performed some additional tests where they compared generalization performance of SGD and heavy-ball SGD against Adagrad, RMSProp, and Adam on known datasets such as image classification using CIFAR-10 [37]. They found that the adaptive methods did not provide any advantage in the generalization performance when compared to the non-adaptive algorithms. This indicates that there exists cases where the non-adaptive methods output solutions that may generalize better than their adaptive counterparts.

Due the the results above, many researchers have attempted to close the generalization gap presented. The work by Keskar and Socher [35] attempts to combine the best of both worlds. Namely, they propose to start training using Adam to reap the rewards of a faster training rate, then swap to SGD to close the generalization gap of [57]. They dub this process by **Sw**itching from **A**dam **t**o **S**GD (SWATS) which is designed such that the switch is automatic, and does not require additional hyperparameters. The way SWATS achieves this is by allowing the switch over point and the step size of SGD to be learned during training. Briefly, the switch over happens by monitoring a value $\gamma_k \in \mathbb{R}$ such that the orthogonal projection of the regular SGD update with respect to γ_k, $-\gamma_k \mathbf{g}_k$ (where $\mathbf{g}_k$ is a stochastic gradient), equals the Adam update, (denoted by $\mathbf{p}_k$) at that same iteration, i.e. $\mathrm{Proj}_{-\gamma_k \mathbf{g}_k} \mathbf{p}_k = \mathbf{p}_k$. When γ_k no longer varies from each iteration, it triggers the switch over and an exponentially weighted moving average of γ_k with decay parameter being Adam's β_2 parameter (see section 4.4) is invoked as the learning rate for SGD. The authors show that in experiments similar to those performed in [57] where Adam does worse than SGD, SWATS performs similar to SGD.

We must be careful, however, as there are applications that have shown greater generality success when it comes to adaptive methods. Zhang et al. [64] talk about the successes seen training attention models (e.g. BERT [16]) with adaptive methods, when compared to SGD, and study why this is the case. Their study arises from the fact that the distribution of stochastic gradients of attention models are heavy-tailed; This leads the authors to pose the question of whether adaptive models are able to better stabilize optimization under heavy-tailed stochastic gradient noise. The authors experimentally show that tasks with heavy-tail distributed stochastic gradients, such as BERT pre-training, will yield outcomes where Adam outperforms SGD, while on tasks where SGD outperforms Adam, the distribution of the stochastic gradients are well concentrated around the mean. The bad performance from SGD can intuitively be described by the fact that these heavy-tailed stochastic gradients will have a strong influence in SGDs trajectory leading to poor performance.

The authors attempt to reduce this influence by considering SGD with gradient clipping. Gradient clipping refers to limiting the norm of a gradient to be contained within a certain range. This technique does not close the gap between Adam and SGD however; The authors thus additionally propose a novel algorithm called **A**daptive **C**oordinate-wise **Clip**ping (ACClip). This algorithm performs adaptive gradient clipping on a coordinate level, and was experimentally shown to outperform Adam on BERT related tasks.

Chapter 3

Background

3.1 Deep Learning Basics

3.1.1 Neural Networks

Inspired by the neuron structure of the brain, *neural networks* (NN) (once called artificial neural networks (ANN)) are highly parameterized machine learning models used to find patterns in data. In other words, they are functions that use data in order to approximate other functions such as the data's distribution function or a function relating the parameters of an observation from the data. One can think of a NN as an interconnected graph where the vertices are nodes.

A key note to mention here is that NNs are not a specific model but a class of models. Most NNs can be split further into separate classes. Two large classes of NNs which encapsulate many models are feed forward neural networks (FFNN), and recurrent neural networks (RNN). FFNNs are a common form of NN in which the data we feed into the algorithm will flow forward through the network without cycling back to previous nodes or layers. RNNs are are like FFNNs but with the additional ability to cycle back to previous nodes and/or layers. A common yet simple example of an FFNN is the perceptron [42] and its multi-layered extension known as the multi-layer perceptron (MLP).

Perceptron and Multi-layer Perceptron

The perceptron is the simplest example of a feed forward neural network. It is comprised of a single layer of nodes called *threshold logic units* (TLU) (see Figure (3.1) **Left** and

Middle) where each input is connected to every TLU. Such a connection configuration is known as a dense layer. The TLU performs two steps; first, it computes a weighted sum of the inputs, then it applies an almost everywhere differentiable, non-linear function, to the entries of the weighted sum of the inputs. This function is referred to as an activation function. The activation function defines the output of the node. For a perceptron, one uses step functions such as the Heavyside step function (3.1), or the sign function (3.2) as its activation function:

Heaviside step function:

$$H(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases} \tag{3.1}$$

Sign function:

$$sgn(x) = \begin{cases} 1 & x > 0 \\ -1 & x \leq 0 \end{cases} \tag{3.2}$$

which returns either 0 or 1, or -1 or 1, respectively, depending on a threshold (here the threshold $\delta = 0$). Using a single TLU, one can interpret the perceptron as a linear binary classifier as it computes a linear combination of the inputs and classifies the observation based on whether the weighted sum is greater than a threshold δ.

Given an input vector $\mathbf{x} \in \mathbb{R}^d$, an associated weight vector $\mathbf{w} \in \mathbb{R}^d$, a bias value $b \in \mathbb{R}$, and an activation function $\sigma : \mathbb{R} \to \mathbb{R}$ we can represent the (single TLU) perceptron as the function $f_{\mathbf{w},b} : \mathbb{R}^d \to \mathbb{R}$ described by

$$f_{\mathbf{w},b}(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + b) \tag{3.3}$$

The bias term, b in (3.3) is an additional component that allows one to shift the activation function. This allows one to have an extra degree of freedom when searching for an optimal solution. A basic, but common, initialization for b is $b = 1$. We note that, like $\mathbf{w}$, the bias, b, also updates during training.

With additional TLUs, the perceptron can be treated as a multiple linear classifier. In general, given a perceptron with K TLUs, we have that the perceptron can be written as the function $f_{\mathbf{W},\mathbf{b}}(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}^K$ described by

$$f_{\mathbf{W},\mathbf{b}}(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}). \tag{3.4}$$

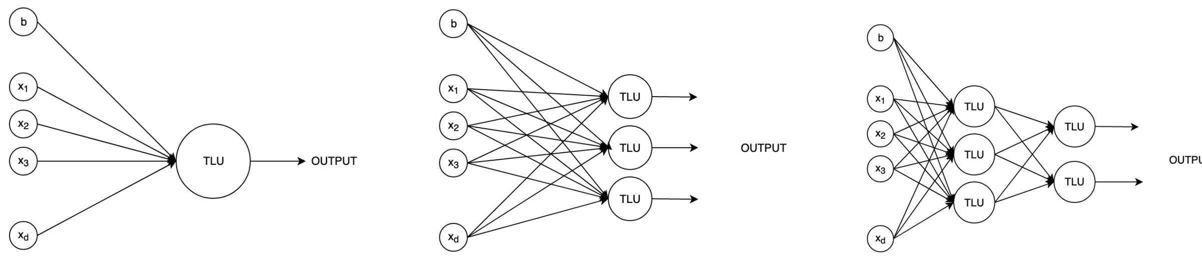


Figure 3.1: **Left:** Single TLU perceptron. **Middle:** Perceptron. **Right:** Multi-layer perceptron.

where $\mathbf{W} \in \mathbb{R}^{K \times d}$, $\mathbf{b} \in \mathbb{R}^{K}$, the vector of 1's, and σ is applied entry-wise. Unlike logistic regression which returns a class probability, the perceptron returns either a "is of positive class" or "is of negative class" style output. The question becomes, how does one optimize the choice of weights $\mathbf{W}$ such that the perceptron can accurately perform classification? For simplicity let us focus on the single-TLU perceptron described in (3.3) with sign activation function (3.2).

Let $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)^\top \in \mathbb{R}^{n \times d}$, so that each observation $\mathbf{x}_i$ has d parameters, $i = 1, \ldots, n$, be the matrix of observations. Let $\mathbf{y} = \{1, -1\}^n \in \mathbb{R}^n$ be the vector of true outcomes (also called labels). Then define the dataset $\mathcal{D} = (\mathbf{X}, \mathbf{y})$ to be the matrix in $\mathbb{R}^{n \times (d+1)}$ in which $\mathbf{y}$ is concatenated column-wise to the end of $\mathbf{X}$. Then we can think of each row of $\mathcal{D}$ to be an observation-label pair, and our goal would be to use the observation to construct a function such that, when given such an input, it can determine the output label. Since we only have one TLU, let $\mathbf{w} \in \mathbb{R}^d$ be the vector of weights such that the j-th coordinate of $\mathbf{w}$, w_j, represents the weight assigned to the j-th parameter x_{ij} of observation $\mathbf{x}_i$, for all $j = 1, \ldots, d$, and all $i = 1, \ldots, n$. Finally, let $b = 1$ be the bias term. The goal would be to minimize the following problem

$$\min_{\mathbf{w} \in R^d,\, b \in \mathbb{R}} \mathcal{L}(\mathcal{D}, \mathbf{w}, b). \tag{3.5}$$

where $\mathcal{L}(\mathcal{D}, \mathbf{w}, b)$ is the perceptron loss function represented by

$$\mathcal{L}(\mathcal{D}, \mathbf{w}, b) = \sum_{i=1}^{n} \max\{0, -\mathbf{y}_i \cdot sgn(\mathbf{w}^\top \mathbf{x}_i + b)\}. \tag{3.6}$$

If $\mathbf{w}$ is chosen such that the sign of the weighted sum matches the label, then we have 0 error for that observation. By taking the sum in (3.6) we are considering every input in $\mathbf{X}$. This particular problem was solved by Rosenblatt using the following algorithm [49, 61].

Algorithm 1 Perceptron Algorithm

Input: $\mathcal{D} = (\mathbf{X}, \mathbf{y})$; where $\mathbf{X} \in \mathbb{R}^{n \times d}$ and $\mathbf{y} \in \mathbb{R}^{\mathbf{n}}$

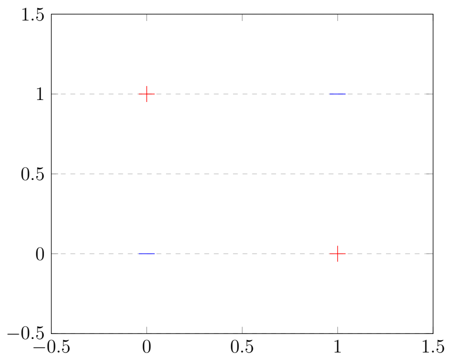
```
Initialize: $\mathbf{w} \in \mathbb{R}^d$, $b \in \mathbb{R}$, $\delta \geq 0$

for $t = 1, 2, \ldots$ do
    Get index $i \in \{1, \ldots, n\}$
    if $\mathbf{y}_i(\mathbf{w}_i^\top \mathbf{x}_i + b) \leq \delta$ then
        $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{y}_i \mathbf{x}_i$
        $b \leftarrow b + \mathbf{y}_i$
    end if
end for
```

Since our activation function is the sign function, we choose $\delta = 0$ in Algorithm (1).

The perceptron has its limitations however. A classic example was shown by Minsky and Papert [42] that a perceptron cannot properly classify the XOR dataset. The XOR data set and a picture of its plot is presented in Figure 3.2. The issue here is that the perceptron outputs a single line to classify inputs. That is, an observation will be classified based on which side of the optimal line it lands in. However, the XOR dataset is designed such that its outputs cannot be separated by any line. This is known as the data being not linearly separable. That is, perceptrons can only classify linearly separable data. However, a multi-layer perceptron can correctly classify the XOR dataset.

	$\mathbf{x_1}$	$\mathbf{x_2}$	$\mathbf{x_3}$	$\mathbf{x_4}$
	0	1	0	0
	0	0	1	1
$\mathbf{y}$	-	+	+	-

(a): XOR dataset



(b): Plot of XOR dataset

Figure 3.2: **(a)** The XOR dataset, and **(b)** its plot.

A multi-layer perceptron can be thought of as a perceptron but with multiple layers (see Figure (3.1) **Right**). Consider an MLP with L layers. Let $\mathbf{X} \in \mathbb{R}^{d_0 \times d_1}$ be the matrix of inputs, where we denote d_0 as the number of inputs and d_i to be the number of parameters of layer $i = 1, \ldots L$. Let $\mathbf{b}_i \in \mathbb{R}^{d_{i+1}}$ be the bias vector for layer i, and let $\mathbf{W}_i \in \mathbb{R}^{d_{i+1} \times d_i}$ be the weight matrix for layer i. Finally, let $\sigma_i : \mathbb{R} \to \mathbb{R}$ be the activation function of layer i. Then we may represent this MLP by the function $f_{MLP}(\mathbf{x}) : \mathbb{R}^{d_1} \to \mathbb{R}^{d_L}$ as

$$f_{MLP}(\mathbf{x}_i) = \mathbf{W}_L \sigma_L(...\sigma_3(\mathbf{W}_2 \sigma_2(\mathbf{W}_1 \mathbf{x}_i + \mathbf{b}_1) + \mathbf{b}_2)...) + \mathbf{b}_L \tag{3.7}$$

again, where σ_i is applied element-wise. We note that one can again apply an activation function to the output of f_{MLP} depending on the desired task as described just below. We note here that in this definition of MLP, we designate a bias value to each node per layer.

While MLPs are more flexible in their abilities compared to a perceptron, the ability to train them requires a more complex approach which will be briefly discussed at the end of this section.

Depending on how one structures the output, MLPs can be used for both regression and classification tasks. If one chooses to perform regression, then it is necessary to have no restriction on our output and thus require that no activation function be used on the output nodes. This allows an output to be of any value. For such tasks, a potential choice of loss function to find the optimal weights and biases is the mean squared error loss

$$\mathcal{L}_{reg}(\mathcal{D}, \mathbf{W}, \mathbf{b}) = \sum_{i=1}^{n} (y_i - f_{MLP}(\mathbf{x}_i))^2. \tag{3.8}$$

However, if one is dealing with a scenario in which output does need to be controlled, say non-negative outputs only, then there exists a handful of activation functions to apply to f_{MLP} such as the rectified linear unit (ReLU) function [21] presented in (3.9)

$$ReLU(x) = \max\{0, x\}. \tag{3.9}$$

For classification tasks, the choice of binary or multi-class classification boils down to the number of output nodes you have as well as a choosing a suitable activation for the output nodes to allow for a representation that signifies a classification. For example, if we wish to perform a binary classification, we require to set the count of the final output nodes to one and choose an activation function that bounds the output between two values such as the sign function in (3.2) or if one wishes to have a probabilistic output then the sigmoid activation (3.10) can be used

$$\phi(x) = \frac{1}{1 + e^{-x}}. \tag{3.10}$$

The sigmoid function restricts the output to be between 0 and 1. The output $\phi(x)$ could be seen as a probability of the input being part of the positive class. If the problem contains multiple binary classifications, also called multi-label binary classification, and would like the MLP to, for example, classify whether a vegetable is green or not, and whether said vegetable is expired or not, then, depending on how many binary labelled classifications are needed, the number of output nodes of the MLP should match the number of said labels. Using the sigmoid activation here, each output would get the probability of being the positive class of that specific binary label choice. Using the vegetable example, the MLP would have two output nodes with the sigmoid activation applied to them. The first output node may output a high probability for "is green", and output a high probability for "not expired" for the second node, yielding the final result of non-expired green vegetable.

As was mentioned previously, in order to train an MLP (and NNs in general), one requires a different approach to that used with the perceptron. The main way in which researchers achieved training NNs came from the breakthrough training algorithm by Rumelheart, Hinton, and Williams known as back propagation [51]. At a high level, back propagation is a method to minimize the loss function by calculating the gradient of the loss function of a NN and performing gradient descent using such gradients. While this sounds simple, given a deep NN with many parameters which depend on one another, back propagation enables one to find the gradient of said loss function with respect to all the parameters in an efficient manner. By efficient, we mean that it requires only "two passes" of the network in order to calculate the gradient of the loss function. The way it does this is via reverse-mode automatic differentiation. Other gradient-based optimizers may be used as well. Stochastic gradient descent and its (adaptive) variants, which are the focus of this thesis, are among the current top choices to use when using back propagation to optimize the loss. Both gradient descent and its stochastic counterpart are discussed next in Section 2.2 which are the foundational building blocks for the adaptive methods introduced and discussed in Chapter 3.

3.2 Gradient-Based Optimization Methods

In this section, we cover the two fundamental optimization methods used to train most machine and deep learning models. As was mentioned in the previous section, the main way to train neural networks is through the use of back propagation. We saw that back propagation is broken down into two parts, the first phase was to find the gradients of the loss function with respect to each weight parameter using reverse-mode automatic differentiation, then perform a correcting step via gradient descent. In this section, we will now explain both gradient descent and, its more popular variant, stochastic gradient descent.

Unless specified, let X denote an Euclidean space with inner product $\langle \cdot\ ,\ \cdot \rangle$, and induced norm $\| \cdot \| = \sqrt{\langle \cdot\ ,\ \cdot \rangle}$. Primarily we will be working with $X = \mathbb{R}^d$ such that $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^\top \mathbf{y}$ for $\mathbf{x}, \mathbf{y} \in X$.

3.2.1 Gradient Descent

Gradient descent (GD) is a type of optimization algorithm known as a first-order iterative method. First-order comes from the fact that gradient descent uses first derivatives as information to optimize an objective, and iterative comes from the idea that one starts with an initial starting point and progressively the algorithm will generate new approximations to the solution using the previous point, that is, the $(n+1)^{th}$ iteration is generated using the n^{th} iteration, and the initial value is our starting point just mentioned. The idea behind gradient descent can best be described by the following analogy. Assume it is a foggy day and a person is on a hill. This person wishes to find the bottom of the hill so that they can go home. Since it is foggy, they cannot see the entire hill around them, only a small radius around them. Then, to get down, this person decides to walk down the path of steepest descent with the hope that they will eventually reach the bottom. Taking the hill to be the objective function, and the bottom of the hill to be the minimum of said function, gradient descent finds minima by traveling the path of steepest descent.

Definition 3.2.1 (Convex Set)**.** Let $C \subset \mathbb{R}^d$. Then C is convex if, for $\forall \lambda \in (0,1)$, $\mathbf{x}, \mathbf{y} \in C$,

$$\lambda \mathbf{x} + (1-\lambda)\mathbf{y} \in C. \tag{3.11}$$

Definition 3.2.2 (Convex Function)**.** Let $f : \mathbb{R}^d \to (-\infty, +\infty]$ be an extended real-valued function. Then f is convex if, $\forall \lambda \in (0,1)$, $\forall \mathbf{x}, \mathbf{y} \in \mathrm{dom} f$,

$$f(\lambda \mathbf{x} + (1-\lambda)\mathbf{y}) \leq \lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{y}). \tag{3.12}$$

where the effective domain of f, $\mathrm{dom} f = \{\mathbf{x} \in X \mid f(\mathbf{x}) < +\infty\}$, is a convex set.

Remark. If we change the inequality in (3.12) to be strict, then we say f is *strictly* convex.

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a continuously differentiable function, and define the following minimization problem

$$\min_{\mathbf{w} \in \mathbb{R}^d} f(\mathbf{w}) \tag{3.13}$$

which is called a smooth unconstrained problem since the domain $\{\mathbf{w} \mid \mathbf{w} \in \mathbb{R}^d\}$ is unconstrained and f is smooth by definition. In general, f need not be convex, however, if it is, we gain the benefit that any local minimizer is automatically a global minimizer:

Theorem 3.2.1. *Let $f : \mathbb{R}^d \to \mathbb{R}$ be convex. If $\mathbf{x} \in X$ is a local minimizer of f, then it is a global minimizer of f.*

Proof. Assume, for eventual contradiction, that $\mathbf{x}$ is a local, but not the global, minimizer for a convex function f. Then there exists a point $\mathbf{y}$ satisfying $f(\mathbf{y}) < f(\mathbf{x})$. However, by convexity of f, for all $\lambda \in (0, 1)$, we get that

$$f(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}) \leq \lambda f(\mathbf{x}) + (1-\lambda)f(\mathbf{y}) < f(\mathbf{x}) \tag{3.14}$$

that is, we may always find a point $\mathbf{z}$ such that, for any λ-ball around $\mathbf{x}$, we have $f(\mathbf{z}) < f(\mathbf{x})$, but this contradicts $\mathbf{x}$ being a local minimizer of f. This concludes the proof. $\square$

A simple example that satisfies the assumptions of f being smooth and convex is the quadratic function $f : \mathbb{R} \to \mathbb{R}$ s.t $f(x) = x^2$. By inspection of the graph (see Figure (3.3) **c)**), the global minimum is at $x = 0$. The question becomes, how does gradient descent find this minimum? To answer this we need to inspect the gradient descent [8, Sect. 9.3] algorithm (2):

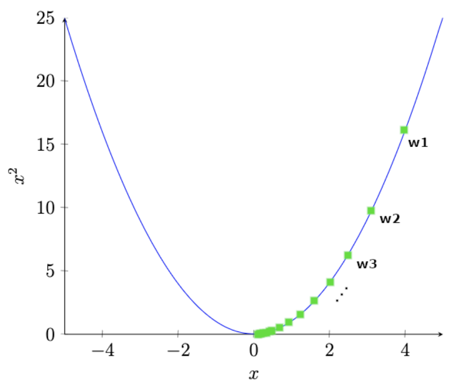
Algorithm 2 Gradient Descent

Input: initial value $\mathbf{w}_0 \in \mathbb{R}^d$, continuously differentiable function $f : \mathbb{R}^d \to \mathbb{R}$
Output: optimal solution $w^* \in \mathbb{R}^d$

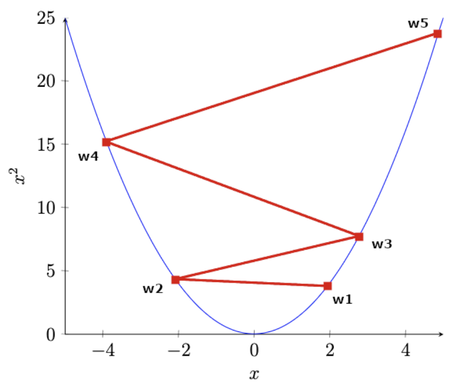
```
1: for $t = 0, 1, 2, \ldots$ do
2:     $\mathbf{g}_t \leftarrow \nabla f(\mathbf{w}_t)$
3:     choose step-size $\eta_t$
4:     $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \mathbf{g}_t$
5: end for
```

For each iteration t in algorithm (2), we compute the gradient of f at the previously computed value $\mathbf{w}_t$ (when t=0, it uses the inputted initial value $\mathbf{w}_0$). The gradient is the direction of steepest ascent, so to "move down the hill" we update our location $\mathbf{w}_{t+1}$ by moving in the negative direction (opposite) of the gradient, i.e. the direction of steepest descent. To see that we are, indeed, decreasing f as we move in this direction, we can use the first-order Taylor's approximation and expand $f(\mathbf{w}_{t+1})$ about $\mathbf{w}_t$ as

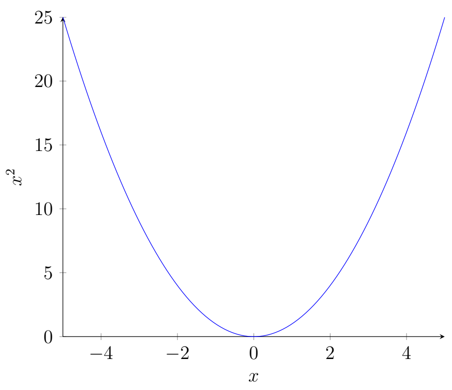
$$\begin{aligned} f(\mathbf{w}_{t+1}) &= f(\mathbf{w}_t - \eta_t \nabla f(\mathbf{w}_t)) \\ &\approx f(\mathbf{w}_t) + \nabla f(\mathbf{w}_t)^\top (\mathbf{w}_t - \eta_t \nabla f(\mathbf{w}_t) - \mathbf{w}_t) \\ &= f(\mathbf{w}_t) - \eta_t \nabla f(\mathbf{w}_t)^\top \nabla f(\mathbf{w}_t) \\ &= f(\mathbf{w}_t) - \eta_t \|\nabla f(\mathbf{w}_t)\|_2^2 \end{aligned} \tag{3.15}$$



a) Iterates of convergent GD on $f(x) = x^2$



b) Iterates of divergent GD on $f(x) = x^2$



c) Graph of $f(x) = x^2$

Figure 3.3: **(c)** Graph of the function $f(x) = x^2$, **(a)** convergent iterates of gradient descent (green) when step size η_t is chosen small enough, and **(b)** divergent iterates of gradient descent (red) when step size η_t chosen too large.

and so for $\eta_t > 0$ small enough, and $\mathbf{w}_t$ not a critical point, we have that $f(\mathbf{w}_{t+1}) < f(\mathbf{w}_t)$, since $\|\nabla f(\mathbf{w}_t)\|_2^2 > 0$. The parameter η_t is called a step-size. It determines how much we move in such a direction and is important for reaching convergence. From our analogy, we can think of the step-size as the radius of vision we have for every step we take. The choice of η_t will be discussed in a moment.

Now, if we have a candidate minimizer $\mathbf{x}^*$ of an objective f, what necessary conditions must this minimizer have? That is, given any minimizer, what properties must they consistently hold? The following theorem tells us that, when we are working in Euclidean space, if a point $\mathbf{x}^*$ is a minimizer of f then the gradient at $\mathbf{x}^*$ must be zero. This is known as Fermat's theorem.

Theorem 3.2.2 (Fermat's Theorem)**.** *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a differentiable function. If $\mathbf{x}^* \in \mathbb{R}^d$ is a local extremum of f, then $\nabla f(\mathbf{x}^*) = \mathbf{0}$.*

Proof. See [60]. $\square$

Remark. The domain of f in Theorem (3.2.2) is specifically $\mathbb{R}^d$ and not a general vector space X.

We must be careful, however. Theorem (3.2.2) is a necessary condition and therefore the converse is not true in general. It is possible to have objective functions such that there exists points $\mathbf{x}$ satisfying $\nabla f(\mathbf{x}) = \mathbf{0}$ but are not extrema of f. Take, for example, the function $f(x, y) = \frac{1}{2}x^2 + \frac{1}{4}y^4 - \frac{1}{2}y^2$; this function has a saddle point at $(0, 0)$, and two global minima at $(0, 1)$ and $(0, -1)$. If we choose our initial point to be $(x_0, 0)$ for any x_0, then our updated points (via GD) would be of the form $(x_t, 0)$. This will converge to $(0, 0)$ which has $\nabla f(0, 0) = (0, 0)$, but is not a minima of f.

In order for the converse to be true, in other words $\nabla f(\mathbf{x}) = \mathbf{0} \implies \mathbf{x}$ is an extremum, we require the additional constraint that f be convex. Going back to our example case: $f(x) = x^2$, we have that f is smooth and convex, therefore if we can find a point x^* satisfying $\nabla f(x^*) = 0$, then x^* is a minimizer of f. Under a good choice of η_t, Figure (3.3) **a)** shows us iterates of GD which converge to the point $x^* = 0$. Since f is smooth and convex, and $\nabla f(x^*) = 0$, we can conclude that $x^* = 0$ is a minimizer for f. Furthermore, by Theorem (3.2.1), x^* must be the global minimizer for f. This shows the advantage of working with convex objective functions.

So far, we have omitted details on the choice of step size η_t. The choice of step size is important because improper selection of the step size can lead to divergence when in actuality, convergence was possible. To see this in action, observe Figure (3.3) **b)**. If we step too far in the direction of steepest descent, we may end up jumping to a new point with higher gradient value. This can cause a chain reaction and cause gradient descent to diverge via exploding gradient values (this term is similar to that seen for back propagation, however, the cause is different).

To properly choose a step size, there exists many techniques, some common methods we explain here. First we have constant step size, that is, $\forall t$, $\eta_t = \eta > 0$. Constant step size is the most practical but to choose a value of η which satisfies descent for each iteration is hard to find. There exists a choice to guarantee convergence which will be shown soon. Another method is to adaptively adjust the step size. exact line search is an example of an adaptive step size selector. In short, for iteration t, we choose $\eta_t = \arg\min_{\alpha>0} f(\mathbf{x}_t - \alpha \cdot \nabla f(\mathbf{x}_t))$. In general, it is not possible to get an exact minimizer η which satisfies the minimization, and therefore this technique is not widely used in practice. Backtracking line search [3] is an alternative popular adaptive method for choosing η_t. Fix $0 < \alpha < 1$, for iteration t, initialize $\eta_t = \frac{1}{2}$. If

$$f(\mathbf{x}_t - \eta_t \nabla f(\mathbf{x}_t)) > f(\mathbf{x}_t) - \frac{\eta_t}{2}\|\nabla f(\mathbf{x}_t)\|_2^2 \tag{3.16}$$

then update $\eta_t = \alpha \cdot \eta_t$ and repeat until (3.16) is no longer satisfied, i.e. when the inequality sign becomes equal or flips. To see which value of η to choose if using the constant rule, or why we use (3.16) in order to find a particular η_t for backtracking requires us to dive into the convergence analysis of gradient descent which we do now.

In order to tackle the convergence analysis of gradient descent, we introduce a few definitions and results:

Definition 3.2.3 (Lipschitz Continuous)**.** Let (X, d_X) and (Y, d_Y) be metric spaces. Let $T : (X, d_X) \to (Y, d_Y)$ be a function from X to Y. Then T is Lipschitz continuous if $\exists L > 0$ such that $\forall \mathbf{x}_1, \mathbf{x}_2 \in X$, T satisfies

$$d_Y(T\mathbf{x}_1 - T\mathbf{x}_2) \leq L \cdot d_X(\mathbf{x}_1 - \mathbf{x}_2). \tag{3.17}$$

For example, if $T : (\mathbb{R}^d, \|\cdot\|_2) \to (\mathbb{R}, |\cdot|)$ is a real-valued function, then T is L-Lipschitz continuous if, $\forall \mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^d$, T satisfies

$$|T\mathbf{x}_1 - T\mathbf{x}_2| \leq L \cdot \|\mathbf{x}_1 - \mathbf{x}_2\|_2.$$

Theorem 3.2.3 (Lipschitz Continuous $\iff$ Bounded Derivative)**.** *Let $T : (\mathbb{R}^d, \|\cdot\|_{(1)}) \to (\mathbb{R}^m, \|\cdot\|_{(2)})$ be differentiable. Then T is L-Lipschitz continuous if and only if $\forall \mathbf{x} \in \mathbb{R}^d$, $\|T'\mathbf{x}\|_{op} \leq L$, where $\|\cdot\|_{op}$ is the operator norm defined by*

$$\|T'\mathbf{x}\|_{op} = \sup_{\|x\|_{(1)} \leq 1} \|T'\mathbf{x}\|_{(2)}.$$

where T' is the derivative/gradient of T.

Proof. See [61]. ☐

Definition 3.2.4 (L-Smooth)**.** Let $T : (\mathbb{R}^d, \|\cdot\|_{(1)}) \to (\mathbb{R}^m, \|\cdot\|_{(2)})$ be differentiable. Then T is called L-smooth $(L > 0)$ if, $\forall \mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^d$, it satisfies

$$\|T'\mathbf{x}_1 - T'\mathbf{x}_2\|_{(M)} \leq L \cdot \|\mathbf{x}_1 - \mathbf{x}_2\|_{(1)} \tag{3.18}$$

where $\|\cdot\|_{(M)}$ is a matrix norm, and T' denotes the derivative/gradient of T.

For example, given a differentiable real-valued function $f : (\mathbb{R}^d, \|\cdot\|_2) \to (\mathbb{R}, |\cdot|)$, f is L-smooth if, $\forall \mathbf{x}_1, \mathbf{x}_2 \in R^d$, f satisfies

$$\|\nabla f(\mathbf{x}_1) - \nabla f(\mathbf{x}_2)\|_2 \leq L \cdot \|\mathbf{x}_1 - \mathbf{x}_2\|_2.$$

A function which is L-smooth is one that guarantees that the difference in gradients between two points is no larger than the difference between the two points themselves, up to a constant factor L. By Theorem (3.2.3) it is easy to see that if a twice-differentiable function f is L-smooth, then it has a bounded second derivative / Hessian. We'll see soon that the update to gradient descent depends on the rate of change of the gradient; Under a L-smooth function, this rate is bounded because the rate of change of the gradient is the Hessian. We now prove the convergence of gradient descent when f is convex differentiable smooth, and η_t is fixed, i.e. $\forall t,\ \eta_t = \eta > 0$.

Theorem 3.2.4 (Convergence of Gradient Descent for L-Smooth, Convex, Differentiable Functions)**.** *Let $f : \mathbb{R}^d \to \mathbb{R}$ be convex, differentiable, and L-smooth for some $L > 0$. Then gradient descent with fixed step size $\eta \in (0, 1/L]$ satisfies*

$$f(\mathbf{x}_t) - f(\mathbf{x}^*) \leq \frac{\|\mathbf{x}_0 - \mathbf{x}^*\|_2^2}{2\eta t}. \tag{3.19}$$

This theorem tells us that, under the given assumptions, gradient descent will converge at a rate of $O(1/t)$, in fact this convergence rate is tight [12]. We briefly mention that Theorem 3.2.4 pertains to the convergence of the objective function values, however, the iterates of GD also converge. The following proof for Theorem (3.2.4) was taken from [25]. We write out the proof here to utilize certain results to explain the step size methods above.

Proof. Since ∇f is Lipschitz with constant L (f is L-smooth), which means $\nabla^2 f \preceq L \cdot \mathbf{I}$ (Theorem (3.2.3)), we have $\forall \mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^d$

$$\begin{aligned} &\langle\ (\mathbf{x} - \mathbf{y}),\ (\nabla^2 f(\mathbf{z}) - L \cdot \mathbf{I})(\mathbf{x} - \mathbf{y})\ \rangle \leq 0 \\ \implies\ & L\|\mathbf{x} - \mathbf{y}\|_2^2 \geq \langle\ (\mathbf{x} - \mathbf{y}),\ \nabla^2 f(\mathbf{z})(\mathbf{x} - \mathbf{y})\ \rangle \end{aligned}$$

Using Taylor's Remainder Theorem, we have that $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, $\exists \mathbf{z} \in [\mathbf{x}, \mathbf{y}]$ such that

$$\begin{aligned} f(\mathbf{y}) &= f(\mathbf{x}) + \langle \nabla f(\mathbf{x}),\ (\mathbf{y}-\mathbf{x}) \rangle + \frac{1}{2} \langle\ (\mathbf{x}-\mathbf{y}),\ \nabla^2 f(\mathbf{z})(\mathbf{x}-\mathbf{y})\ \rangle \\ &\le f(\mathbf{x}) + \langle \nabla f(\mathbf{x}),\ (\mathbf{y}-\mathbf{x}) \rangle + \frac{L}{2} \|\mathbf{y} - x\|_2^2 \end{aligned} \tag{3.20}$$

plugging in $\mathbf{y} = \mathbf{x}^+ \equiv \mathbf{x} - \eta \nabla f(\mathbf{x})$ we get

$$\begin{aligned} f(\mathbf{x}^+) &\le f(\mathbf{x}) + \langle \nabla f(\mathbf{x}),\ (\mathbf{x} - \eta \nabla f(\mathbf{x}) - \mathbf{x}) \rangle + \frac{L}{2} \|\mathbf{x} - \eta \nabla f(\mathbf{x}) - \mathbf{x}\|_2^2 \\ &= f(\mathbf{x}) - (1 - \frac{L\eta}{2}) \eta \|\nabla f(\mathbf{x})\|_2^2. \end{aligned} \tag{3.21}$$

Taking $0 < \eta \le 1/L$, which implies that $(1 - L\eta/2) \ge 1/2$ we get that

$$f(\mathbf{x}^+) \le f(\mathbf{x}) - \frac{\eta}{2} \|\nabla f(\mathbf{x})\|_2^2 \tag{3.22}$$

since f is convex we have that $f(\mathbf{x}) \le f(\mathbf{x}^*) + \langle \nabla f(\mathbf{x}), (\mathbf{x} - \mathbf{x}^*) \rangle$ and so

$$\begin{aligned} f(\mathbf{x}^+) &\le f(\mathbf{x}) - \frac{\eta}{2} \|\nabla f(\mathbf{x})\|_2^2 \\ &\le f(\mathbf{x}^*) + \langle \nabla f(\mathbf{x}), (\mathbf{x} - \mathbf{x}^*) \rangle \\ &= f(\mathbf{x}^*) + \frac{1}{2\eta} (\|\mathbf{x} - \mathbf{x}^*\|_2^2 - \|\mathbf{x} - \mathbf{x}^* - \eta \nabla f(\mathbf{x})\|_2^2) \\ &= f(\mathbf{x}^*) + \frac{1}{2\eta} (\|\mathbf{x} - \mathbf{x}^*\|_2^2 - \|\mathbf{x}^+ - \mathbf{x}^*\|_2^2). \end{aligned} \tag{3.23}$$

Summing over iterations, we have

$$\sum_{i=1}^{t} (f(\mathbf{x}_t) - f(\mathbf{x}^*)) \le \frac{1}{2\eta} (\|\mathbf{x}_0 - \mathbf{x}^*\|_2^2 - \|\mathbf{x}_t - \mathbf{x}^*\|_2^2) \tag{3.24}$$

$$\le \frac{1}{2\eta} \|\mathbf{x}_0 - \mathbf{x}^*\|_2^2 \tag{3.25}$$

by (3.22), $f(\mathbf{x}_t)$ is non-increasing therefore we have

$$f(\mathbf{x}_t) - f(\mathbf{x}^*) \le \frac{1}{t} \sum_{i=1}^{t} (f(\mathbf{x}_t) - f(\mathbf{x}^*)) \le \frac{\|\mathbf{x}_0 - \mathbf{x}^*\|_2^2}{2\eta t}. \tag{3.26}$$

This concludes the proof. □

Coming back to the question of constant step size selection, we can use the upper bound in (3.19) in order to derive an η_t. Notice in the proof that we restricted $\eta \in (0, 1/L]$. If we choose $\eta_t \equiv 1/L$, then the upper bound is minimized. That is, under the choice of constant step size $\eta_t \equiv 1/L$, we will get the smallest possible upper bound to (3.19) compared to any other choice in $(0, 1/L]$. If η is chosen to be greater than $2/L$, then the proof can no longer guarantee convergence.

As for the backtracking method, notice equation (3.21) in the proof of Theorem (3.2.4). Since $L > 0$, we have that $0 < (1 - L\eta/2) < 1$, and this value is multiplied by η to satisfy the inequality. That is, there exists some $\beta^* \in (0, 1)$ $(\equiv (1 - L\eta/2))$ such that $b^* \cdot \eta$ will satisfy (3.21) given $\eta \in (0, 1/L]$. This is where (3.16) comes from. We do not know (or have to know) L, we can just pick some $\beta \in (0, 1)$ and iteratively update η by $\beta \cdot \eta$ until the inequality in (3.16) satisfies

$$f(\mathbf{x}_t - \eta \nabla f(\mathbf{x}_t)) \leq f(\mathbf{x}_t) - \eta \|\nabla f(\mathbf{x}_t)\|_2^2. \tag{3.27}$$

This idea of starting η at $1/2$ and iteratively reducing the value by a factor of β until we satisfy (3.27) is where the name "backtracking" comes from. Recommended choices of β are between 0.1 and 0.8 [8].

As was concluded by Theorem (3.2.4), the rate of convergence of gradient descent for L-smooth convex functions with constant step size was $O(1/t)$ where t is the number of iterations. This rate is known as *sub-linear* convergence which we define now

Definition 3.2.5 (Sub-Linear Convergence)**.** Let $\{a_n\}_{n=m}^{\infty}$ be a positive sequence such that $\lim_{n\to\infty} a_n = 0$, and

$$\lim_{n\to\infty} \frac{a_{n+1}}{a_n} = C \tag{3.28}$$

for some $C > 0$. If $C = 1$, then $\{a_n\}_{n=m}^{\infty}$ is said to converge sub-linearly.

By the definition, we can easily conclude sub-linear convergence of gradient descent under Theorem (3.2.4) since

$$\lim_{t\to\infty} \frac{1/(t+1)}{1/t} = \lim_{t\to\infty} \frac{t}{t+1} = 1.$$

where $a_t = 1/t$ comes from the convergence rate upper bound $O(1/t)$. Sub-linear convergence is considered slow. The reason for this is that the computational time to compute a correct new right digit of the solution is similar to the total amount of time taken of all previous work done [44].

It is possible to improve the convergence rate for gradient descent if we tack on the constraint that the objective function we are minimizing is also strongly convex:

Definition 3.2.6 (α-Strongly Convex)**.** Let $f : X \to \mathbb{R}$ be differentiable. Then f is α-strongly convex if, $\forall \mathbf{x}, \mathbf{y} \in X$, it satisfies

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^\top (\mathbf{y} - \mathbf{x}) + \frac{\alpha}{2} \|\mathbf{y} - \mathbf{x}\|^2 \tag{3.29}$$

for some $\alpha > 0$.

Under strong convexity, gradient descent can achieve a linear convergence rate:

Definition 3.2.7 (Linear Convergence)**.** An algorithm is said to converge at a linear rate if, given the sequence of its iterates $\{\mathbf{x}_t\}_{t=0}^{\infty} \to \mathbf{x}^*$, we have that the distance between an iterate $\mathbf{x}_t$ and the optimal solution $\mathbf{x}^*$ is bounded by an exponential of the iteration number

$$\|\mathbf{x}_t - \mathbf{x}^*\| \leq c(1-q)^t \tag{3.30}$$

where $c > 0$, $q \in (0, 1)$.

Theorem 3.2.5 (Convergence of Gradient Descent for L-Smooth, Strongly Convex, Differentiable Functions)**.** *Let $f : \mathbb{R}^d \to \mathbb{R}$ be L-smooth, α-strongly convex, and differentiable. Then gradient descent with fixed step-size $\eta = 1/L$ satisfies*

$$\|\mathbf{x}_t - \mathbf{x}^*\|_2 \leq \left(1 - \frac{\alpha}{L}\right)^t \cdot \|\mathbf{x}_0 - \mathbf{x}^*\|_2. \tag{3.31}$$

Proof. See [33]. □

Setting $q = \alpha/L$, and $c = \|\mathbf{x}_0 - \mathbf{x}^*\|_2$, we see by definition (3.2.7) that GD converges linearly under Theorem (3.2.5).

3.2.2 Stochastic Gradient Descent

In the domain of machine learning, many state of the art models require large amounts of data in order to properly be trained. Due to being in a large data regime, gradient descent becomes a very slow algorithm to use when optimizing because the computational cost of computing the full gradient at each iteration is expensive. To see this problem in action, assume we have a data set with $1'000'000$ data points each with 10 features. That is, our data matrix has dimensions $1'000'000 \times 10$. If our loss function is set to be the mean squared error loss seen in (3.8), then for each gradient computation, we have $n = 1'000'000$ terms each of which need to have their partial derivative taken with respect to the $d = 10$ features. That is, it will take $10 \cdot (1'000'000) = 10'000'000$ computations in order to calculate the full gradient for one iteration. To overcome this, stochastic gradient descent (SGD) tackles this problem by randomly selecting a single data point per iteration in order to calculate the derivative. SGD was first introduced by Robins and Monro [48] as a Markov chain method and not the gradient based method it is today. With that in mind, in this section we'll formally tackle SGD as it is used in machine learning today.

The notion of a loss function was already introduced in the previous sections. Recall that we may write a loss function as the form $\mathcal{L}(\mathcal{D}, \mathbf{w}, \mathbf{b})$, where $\mathcal{D}$ is the data set containing feature-label pairs $(\mathbf{x}, y)$. For more clarity, we rewrite this representation as $\ell(f(\cdot, \mathbf{w}, \mathbf{b}), \mathbf{y})$, where $f(\cdot, \mathbf{w}, \mathbf{b})$ is the predictor function (for example f_{MLP} from (3.7)). It is possible to integrate the bias into the weight vector so we'll write the loss as $\ell(f(\cdot, \mathbf{w}), \mathbf{y})$ for simplicity. This interpretation is more clear because we may now see that the loss is a function of the predictor and true label, and we aim to minimize ℓ by finding the $f(\cdot, \mathbf{w})$ that minimizes it.

Since the goal would be for the model to perform well when presented with unseen data, it is better to find the weights $\mathbf{w}$ which minimize the loss function given those data points. However, when training a model, we don't have this data at hand, only a training sample. To overcome this, we would like to choose $\mathbf{w}$ that minimizes the expected loss for any input-output pair [7]. Formally speaking, our input-output pair can be viewed as random variables $(\mathbf{X}, \mathbf{Y}) \sim P(\mathbf{X} = \mathbf{x}, \mathbf{Y} = \mathbf{y})$ where $P : (\mathbb{R}^{d_x} \times \mathbb{R}^{d_y}) \to [0, 1]$ is the probability distribution of the true $\mathbf{x}$ - $\mathbf{y}$ relationship, and d_x and d_y are the respective dimensions for $\mathbf{x}$ and $\mathbf{y}$. This new objective is known as *expected risk* and is defined as:

(Expected Risk I)

$$R(\mathbf{w}) = \int_{\mathbb{R}^{d_x} \times \mathbb{R}^{d_y}} \ell(f(\mathbf{x}, \mathbf{w}), \mathbf{y}) \, dP(\mathbf{x}, \mathbf{y}) = \mathbb{E}[\ell(f(\mathbf{X}, \mathbf{w}), \mathbf{Y})] \tag{3.32}$$

In practice, it is impractical to minimize (3.32) since this requires access to the entire probability distribution, which is generally not known. Instead, one uses an estimate of R in (3.32) [7]. This estimate is known as *empirical loss* and is defined by:

(Empirical Risk I)

$$R_n(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \ell(f(\mathbf{x}_i, \mathbf{w}), \mathbf{y}_i) \tag{3.33}$$

where n is the number of data points in the training sample.

In order to proceed, we take a moment to simplify the notation of (3.32) and (3.33) to the more common form used in practice. Let ζ be a random variable such that the realization of ζ is either a single input-output sample $(\mathbf{x}_i, \mathbf{y}_i) \in (\mathbb{R}^{d_x} \times \mathbb{R}^{d_y})$ or a set of samples $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i \in \mathcal{S}}$ where $S \subset (\mathbb{R}^{d_x} \times \mathbb{R}^{d_y})$ is a subset of the sample space. Next, we rewrite the loss function as $h(\mathbf{w}; \zeta) = \ell(f(\cdot_x, \mathbf{w}), \cdot_y)$, where $(\cdot_x, \cdot_y) \sim \zeta$. That is, h is the composition of ℓ and f. Then we can rewrite (3.32) as:

(Expected Risk II)

$$R(\mathbf{w}) = \mathbb{E}[h(\mathbf{w}, \zeta)]. \tag{3.34}$$

Continuing from our definition of ζ, we denote $\hat{\zeta}_i$ to be the realization of ζ for a single training sample $(\mathbf{x}_i, \mathbf{y}_i)$, $i \in \{1, \dots, n\}$, from the set of n-realized samples given to ζ. In other words, given a set of n realizations for ζ, $\hat{\zeta}_i$ is the i^{th} element of that set. We can, therefore, represent the set of all realized samples of ζ to be $\{\hat{\zeta}_i\}_{i=1}^n = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$. Given a realized set of samples, $\{\hat{\zeta}_i\}_{i=1}^n$ for the random variable ζ, let $h_i(\mathbf{w})$ denote the loss with respect to the i^{th} realized sample $\hat{\zeta}_i$, i.e. $h_i(\mathbf{w}) = h(\mathbf{w}; \hat{\zeta}_i)$. Then we rewrite (3.33) as:

(Empirical Risk II)

$$R_n(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} h_i(\mathbf{w}). \tag{3.35}$$

Going forward, unless specified, any use of the notation $R(\mathbf{w})$ or $R_n(\mathbf{w})$ refers to (3.34) and (3.35), respectively.

We now present the stochastic gradient descent algorithm. Let $k \in \mathbb{N}$ be such that ζ_k denotes the k^{th} random variable of a sequence of jointly independent random variables $\{\zeta_k\}_{k=0}^{\infty}$. Each ζ_k can be seen as a sample generated from an unknown data distribution. We denote the realization of ζ_k as $\hat{\zeta}_k = (\mathbf{x}_k, \mathbf{y}_k)$ some point from the data distribution.

Algorithm 3 Stochastic Gradient Descent

Input: initial value $\mathbf{w}_0 \in \mathbb{R}^d$

```
1: for $t = 0, 1, 2, \dots$ do
2:     Get realization $\hat{\zeta}_t$ from $\zeta_t$
3:     $\mathbf{g}_t \leftarrow \nabla h_t(\mathbf{w}_t)$  $[\equiv \nabla h(\mathbf{w}_t; \hat{\zeta}_t)]$
4:     choose step-size $\eta_t$
5:     $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \mathbf{g}_t$
6: end for
```

At line 3 of Algorithm (3), we wrote the gradient of a general loss function $h_i(\mathbf{w}) = h(\mathbf{w};\zeta_i)$. With respect to empirical risk (3.35), we would simply change line 3 of Algorithm (3) to be:

$$\mathbf{g}_t \leftarrow \nabla R_n(\mathbf{w}).$$

The following convergence analysis of stochastic gradient descent works for both expected and empirical risk, where the difference comes from how one picks the samples [7]. As such, we denote our objective to be

$$f(\mathbf{w}_t) := \begin{cases} R(\mathbf{w}_t) \\ \text{or} \\ R_n(\mathbf{w}_t) \end{cases} \tag{3.36}$$

and denote the stochastic gradient as

$$\mathbf{g}_t := \nabla h(\mathbf{w}_t;\zeta_t) \tag{3.37}$$

and require the assumptions that our objective is L-smooth, α-strongly convex, and the following additional assumption [7]:

(First and Second Moment Limits)
The objective function, f, and the SGD algorithm must satisfy the following three requirements:

(a) The sequence of iterates $\{\mathbf{w}_t\}$ is contained in an open set over which f is bounded below by a scalar f_{inf}.

(b) There exist scalars μ_G and μ satisfying $\mu_G \geq \mu > 0$, such that, $\forall t \in \mathbb{N}$,

$$\nabla f(\mathbf{w}_t)^\top \mathbb{E}_{\zeta_t}[\mathbf{g}_t] \geq \mu \|\nabla f(\mathbf{w}_t)\|_2^2, \quad \textit{and} \tag{3.38}$$

$$\|\mathbb{E}_{\zeta_t}[\mathbf{g}_t]\|_2 \leq \mu_G \|\nabla f(\mathbf{w}_t)\|_2. \tag{3.39}$$

(c) There exist scalars $M \geq 0$ and $M_V \geq 0$ such that, $\forall t \in \mathbb{N}$,

$$\mathrm{Var}_{\zeta_t}[\mathbf{g}_t] \leq M + M_V \|\nabla f(\mathbf{w}_t)\|_2^2 \tag{3.40}$$

where $\mathrm{Var}_{\zeta_t}[\mathbf{g}_t] = \mathbb{E}_{\zeta_t}[\|\mathbf{g}_t\|_2^2] - \|\mathbb{E}_{\zeta_t}[\mathbf{g}_t]\|_2^2$ is the variance of $\mathbf{g}_t$. *(a)* requires our objective function to be bounded below, *(b)* requires the direction $-\mathbf{g}_t$ to satisfy being a sufficient descent direction of f at the point $\mathbf{w}_t$ in terms of the norm of the full gradient, and *(c)* weakly constrains the variance of $\mathbf{g}_t$.

Theorem 3.2.6. *Let $f : \mathbb{R}^d \to \mathbb{R}$ be L-smooth, α-strongly convex, and differentiable. Additionally, assume the First and Second Moment Limits above with $f_{inf} = f_*$ (f_* is the minimum of f). Furthermore, suppose that SGD is run with a fixed step size $\eta_t \equiv \eta > 0$ for all $t \in \mathbb{N}$ such that η satisfies:*

$$\eta \leq \frac{\mu}{LM_G},$$

where $M_G = M_V + \mu_G^2 \geq \mu^2 > 0$. Then the expected optimality gap satisfies the following inequality for all $t \in \mathbb{N}$:

$$\mathbb{E}[f(\mathbf{w}_t) - f_*] \leq \frac{\eta LM}{2\alpha\mu} + (1 - \eta\alpha\mu)^{t-1} \left(f(\mathbf{w}_1) - f_* - \frac{\eta LM}{2\alpha\mu} \right) \to \frac{\eta LM}{2\alpha\mu} \tag{3.41}$$

as $t \to \infty$.

Proof. See [7]. □

If our direction $\mathbf{g}_t$ is an unbiased estimator of $\nabla f(\mathbf{w}_t)$, i.e. $\mathbb{E}[\mathbf{g}_t] = \nabla f(\mathbf{w}_t)$, and $\mathbf{g}_t$ has no noise, then we get that $\mu = 1$ and $M_G = 1$ [7]. This yields $\eta \in (0, L]$ which was the requirement for η in Theorem (3.2.4).

In practice, it is often not wise to choose a constant step size η. The reason for this is that, while a constant step size can guarantee convergence to a neighborhood of the optimal solution, the noise of the estimated gradient $\mathbf{g}_t$ will prevent any further progress towards the optimum [7]. Inspecting (3.41), it is possible to tighten the optimality gap by reducing the value of η at the cost of a worse contraction constant. One can take advantage of this fact to get the following result:

Theorem 3.2.7. *Let $f : \mathbb{R}^d \to \mathbb{R}$ be L-smooth, α-strongly convex, and differentiable. Additionally, assume the (First and Second Moment Limits) assumptions above with $f_{inf} = f_*$ (f_* is the minimum of f). Furthermore, suppose that SGD is run with a step size sequence, η_t, such that $\forall t \in \mathbb{N}$:*

$$\eta_t = \frac{\beta}{\gamma + t}$$

for some $\beta > 1/(\alpha\mu)$ and $\gamma > 0$ such that $\eta_1 \leq \mu/(LM_G)$. Then, $\forall t \in \mathbb{N}$, the expected optimality gap satisfies

$$\mathbb{E}[f(\mathbf{w}_t) - f_*] \leq \frac{\nu}{\gamma + t} \tag{3.42}$$

where

$$\nu = \max \frac{\beta^2 LM}{2(\beta\alpha\mu - 1)}, (\gamma + 1)(f(\mathbf{w}_1) - f_*). \tag{3.43}$$

Proof. See [7]. □

A strategy that may be used in practice which takes advantage of the diminishing step size effect is as follows: choose a constant step size and run the algorithm until progress is no longer made. When this stall happens, a smaller step size is chosen and the algorithm continues. This process is repeated only when the algorithm stalls. However, it is generally not easy to detect such a "lack of progress".

The assumption of α-strong convexity of the objective function for the convergence of SGD is crucial in that it is a critical component of ensuring a $O(1/t)$ convergence rate. If the strong convexity parameter is incorrectly estimated, even with unbiased $\mathbf{g}_t$ and $\mu = 1$, a solution can fail to close the optimality gap [43]. We further note that, even under these stronger assumptions, SGD converges at a sub-linear rate while GD converges linearly. This is a trade-off between GD and SGD that needs to be made in order to reap the beneficial computational speeds of SGD. It is worth noting that there do exist ways to guarantee a linear convergence rate for SGD if even more information is provided. For example, stochastic variance reduced gradient (SVRG) [32] is an alternative to SGD in which the stochastic gradient is updated in an amortized fashion. This requires, on average, two gradient computations per step, but has the guarantee of linear convergence [58].

3.2.3 A Practical Compromise Between GD and SGD: Mini-Batch Gradient Descent

While it is true that, per iteration, SGD is computationally faster than standard GD, this speed is some-what balanced out by the fact that SGD requires a large number of iterations in order to get close to a minima. This was due to the fact that SGD's update direction is a descent direction on average; it may not be at each iteration. Even then, it was seen that SGD usually fails to converge to a minima, but instead jumps around in a neighborhood of the solution. In contrast, gradient descent is computationally inefficient for large data sets yet satisfies a more direct direction of descent and can converge. In practice, it is more advantageous to choose an option somewhere in between SGD and GD that has trade offs between gradient descent and stochastic gradient descent. This is called *mini-batch gradient descent* (MBGD).

As we saw, SGD randomly picks a single sample point $(\mathbf{x}, \mathbf{y})$ to compute the update. Mini-batch GD does something similar in that, instead of randomly selecting a sample point, we split the entire dataset into approximately equal partitions, then perform an update using a single partition of data which is called a mini-batch. After training on each of the partitions, the process is repeated; one pass through all the mini-batches is called an *epoch*. It is common to iterate through a large number of epochs before attaining convergence to a solution.

As a quick example, say we had a dataset consisting of 100'000 sample pairs $\mathcal{D} = \{(\mathbf{x}, \mathbf{y})\}^{100000}$; a possible partitioning of the dataset into mini-batches would be every 1000 sample points:

$$\begin{aligned} \text{batch}_1 &: \{(\mathbf{x}, \mathbf{y})\}^{[0:1000]} \\ \text{batch}_2 &: \{(\mathbf{x}, \mathbf{y})\}^{[1000:2000]} \\ &\vdots \end{aligned}$$

where the square bracket notation here $[a : b]$ represents taking a slice from sample a to sample $b - 1$ of the dataset.

If we chose our mini-batch sizes to be of size 1, then we retrieve stochastic gradient descent, while on the other extreme, choosing the mini-batch sizes to be the full batch (i.e. one mini-batch contains the entire dataset), then we retrieve gradient descent (also called full-batch gradient descent). In practice, the rule of thumb for choosing a mini-batch size is above 10 to take advantage of the higher computation speed-ups of matrix-matrix product compared to matrix-vector products [6]. It is also recommended to stick with values that are powers of 2 due to potential computational speed ups based on how computer memory is stored and accessed.

To conclude, mini-batch GD is widely used in practice over full-batch GD and stochastic GD. It is a method used to enhance training speed performance; theoretically, the choice of mini-batch size does not affect the model's generalization performance [6].

3.2.4 Subgradient Descent

So far, we've considered the strong assumption of smoothness for our objective functions. In practice however, it is very common that our objective functions don't have the structural properties of differentiability/smoothness. As an example, the ReLU function (3.9) is not differentiable at $x = 0$. How can we deal with this? The answer lies in the use of subgradients.

Recall that the proper domain of a function $f : \mathbb{R}^n \to [-\infty, +\infty]$, denoted $\mathrm{dom}(f)$, is the set:

$$\mathrm{dom}(f) := \{\mathbf{x} \in \mathbb{R}^n \mid f(\mathbf{x}) < +\infty\}.$$

Definition 3.2.8 (Proper Function)**.** The function $f : \mathbb{R}^n \to [-\infty, +\infty]$ is proper if $\mathrm{dom}(f) \neq \emptyset$ and for all $\mathbf{x} \in \mathbb{R}^n$, $f(\mathbf{x}) > -\infty$.

Definition 3.2.9 (Subgradient)**.** Let $f : \mathbb{R}^n \to [-\infty, +\infty]$ be a proper function. The subgradient of f at $\mathbf{x} \in \mathbb{R}^n$ is defined to be a vector $\mathbf{g} \in \mathbb{R}^n$ that satisfies:

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \langle \mathbf{g}, \mathbf{y} - \mathbf{x} \rangle \tag{3.44}$$

for all $\mathbf{y} \in \mathbb{R}^n$.

It is important to note that a subgradient at a point $\mathbf{x}$ may not be unique. We denote the set of vectors that satisfy (3.44) at $\mathbf{x}$ by: $\partial f(\mathbf{x})$, which we can write as:

$$\partial f(\mathbf{x}) := \{\mathbf{g} \in \mathbb{R}^n \mid f(\mathbf{y}) \geq f(\mathbf{x}) + \langle \mathbf{g}, \mathbf{y} - \mathbf{x} \rangle,\ \forall \mathbf{y} \in \mathbb{R}^n\}. \tag{3.45}$$

We call $\partial f(\mathbf{x})$ the subdifferential of f at $\mathbf{x}$. Intuitively, the subdifferential of a function f at the point $\mathbf{x}$ is the set of all vectors $\mathbf{g}$ such that we may construct a linear lower bound for the function f.

As an example, the function $f : \mathbb{R} \to \mathbb{R}$ such that $x \to |x|$ is not differentiable at $x = 0$ and therefore does not have a derivative over $\mathbb{R}$. However, it does have a subgradient which is written as:

$$\partial f(x) = \begin{cases} \{-1\} & x < 0 \\ [-1, 1] & x = 0 \\ \{+1\} & x > 0 \end{cases} \tag{3.46}$$

One result before we get to the main algorithm and analysis results is the analogous statement of Theorem (3.2.3) but for subgradients:

Theorem 3.2.8. *Let $f : \mathbb{R}^n \to (-\infty, +\infty]$ be proper and convex. Then f is L-Lipschitz continuous in the interior of dom(f) (with respect to $\|\cdot\|_2$) if and only if for all $\mathbf{x}$ in the interior dom(f) and $\mathbf{g} \in \partial f(\mathbf{x})$ we have that $\|\mathbf{g}\|_2 \leq L$.*

Proof. See [46]. □

The subgradient descent algorithm is as follows:

Algorithm 4 Subgradient Descent

Input: initial value $\mathbf{w}_0 \in \mathbb{R}^d$
Output: optimal solution $\mathbf{w}^* \in \mathbb{R}^d$

```
1: for $t = 0, 1, 2, \ldots$ do
2:     choose $\mathbf{g}_t \in \partial f(\mathbf{w}_t)$
3:     choose step-size $\eta_t$
4:     $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \mathbf{g}_t$
5: end for
```

As we can see, subgradient descent is similar to gradient descent except we simply substitute the gradient of f with a subgradient of f, at a point $\mathbf{x}$. We now present the convergence analysis results:

Theorem 3.2.9. *Let $f : R^d \to \mathbb{R}$ be proper, convex, and L-Lipschitz continuous with respect to $\| \cdot \|_2$, i.e. $\|\mathbf{g}\|_2 \leq L$, $\forall \mathbf{g} \in \partial f(\mathbf{x})$. Let $\{\mathbf{x}_t\}_{n=1}^{\infty}$ be a sequence generated by Algorithm (4). Then:*

$$\min_{0 \leq t \leq T-1} f(\mathbf{x}_t) - f(\mathbf{x}^*) \leq \frac{\|\mathbf{x}_0 - \mathbf{x}^*\|_2^2 + \sum_{t=0}^{T-1} \eta_i^2 \|\mathbf{g}_t\|_2^2}{2\sum_{t=0}^{T-1} \eta_t} \leq \frac{\|\mathbf{x}_0 - \mathbf{x}^*\|_2^2 + L^2 \sum_{t=0}^{T-1} \eta_i^2}{2\sum_{t=0}^{T-1} \eta_i}. \quad (3.47)$$

Moreover, if we assume, in addition, that f is α-strongly convex (with respect to $\| \cdot \|_2$), and setting $\eta_t = 1/(\alpha(t+1))$ we have:

$$\min_{0 \leq t \leq T-1} f(\mathbf{x}_t) - f(\mathbf{x}^*) \leq \sum_{t=0}^{T-1} \frac{1}{T}(f(\mathbf{x}_t) - f(\mathbf{x}^*)) \leq \frac{L^2 \sum_{t=0}^{T-1} \frac{1}{1+t}}{2\alpha T} \quad (3.48)$$

Proof. See [62]. □

Under the strongly convex case, we have that the right hand side of (3.48) converges to 0 at a rate of $O(\log T/T)$ [62].

3.3 Online (Convex) Learning

As we will come to see later on, the results of the adaptive methods we discuss are catered towards the online convex optimization setting. It is worth noting that this setting is almost equivalent to working in the stochastic setting [13]. To maintain consistency with these results, we give a quick high-level introduction to online (convex) learning.

To better grasp what online learning is, we'll build up the main intuition via an example. The following example and build up to online learning was heavily inspired by the following monograph by Orabona [46]. To begin, consider the following example: you and an adversary are playing a game consisting of $t = 1, 2, \ldots, T$ rounds. In each round, the adversary draws an independent and identically distributed (i.i.d) number y_t constrained to the interval $[0, 1]$. Your goal is to choose an $x_t \in [0, 1]$ such that it minimizes some loss function $\ell_t(x_t; y_t)$, for example the squared loss $\ell(x_t; y_t) = (x_t - y_t)^2$. If the distribution was known before hand, then the best pick would be the expected value of that distribution, and thus, under the squared loss, we'd have an expected loss of $\sigma^2 T$ by the end of the game, where σ^2 is the variance of the distribution that the adversary drew from. In the general case, however, one does not know the distribution. So we aim to pick x_t such that we'd like to minimize the following "regret":

$$\mathbb{E}_Y\left[\sum_{t=1}^{T} \ell(x_t)\right] - \sigma^2 T = \mathbb{E}_Y\left[\sum_{t=1}^{T} (x_t - Y)^2\right] - \sigma^2 T \tag{3.49}$$

or the average:

$$\frac{1}{T}\mathbb{E}_Y\left[\sum_{t=1}^{T} \ell(x_t)\right] - \sigma^2 = \frac{1}{T}\mathbb{E}_Y\left[\sum_{t=1}^{T} (x_t - Y)^2\right] - \sigma^2 \tag{3.50}$$

where $Y \sim \textit{Distribution}$ is a random variable drawn from the unknown distribution. In the online learning world, winning at this game is governed by if you are able to choose x_t such that (3.49) grows sub-linearly over time, or, equivalently, if (3.50) goes to 0 as $T \to \infty$. That is, these are the (equivalent) measures of success.

Instead of being drawn from a distribution, let y_t be an arbitrary sequence of realized values (no longer random nor are each element drawn from the same distribution). In fact, the choice of y_t can be selected adversarially in order to make us "lose". In this regime, we therefore cannot rely on statistical modeling of the data, and the $\sigma^2 T$ is no longer the optimal choice. We therefore rewrite (3.49) as:

$$Regret_T := \sum_{t=1}^{T} l(x_t; y_t) - \min_{x\in[0,1]} \sum_{t=1}^{T} l(x; y_t) \left(= \sum_{t=1}^{T} (x_t - y_t)^2 - \min_{x\in[0,1]} \sum_{t=1}^{T} (x - y_t)^2 \right) \quad (3.51)$$

where we win if (3.51) grows sub-linearly with respect to T. In the literature, (3.51) is called the regret. We now have the foundation to extend this example to a generalized notion.

Let $V \subset \mathbb{R}^n$, called the feasible set, and let $\mathbf{x}_t \in V$. Here $\mathbf{x}_t$ is the output from your model. Attached to this output is a loss function $\ell_t : V \to \mathbb{R}$ that measures how good the output prediction was to the true value. Next, let $\mathbf{u} \in V$ be an arbitrary predictor for the same true value. $\mathbf{u}$ is not outputted by the model, but instead is used to compare against the output $\mathbf{x}_t$ of the model. Then we can write the Regret function as a function of $\mathbf{u}$ as:

$$Regret_T(\mathbf{u}) = \sum_{t=1}^{T} \ell_t(\mathbf{x}_t) - \sum_{t=1}^{T} \ell_t(\mathbf{u}). \quad (3.52)$$

We can see now why the function is called a regret function. We are essentially calculating how much "regret" the model has for choosing its parameter $\mathbf{x}_t$ over the arbitrary one $\mathbf{u}$.

The question becomes, how do we "win" this game? i.e. can we formulate an algorithm such that it guarantees sub-linear regret over T? Of course the answer is yes, and there exists many algorithms out there. We will demonstrate Online Gradient Descent (OGD) as this is the algorithm the adaptive methods aim to improve on. Before we get to the algorithm, we notice that our choice of $\mathbf{x}_t$ is constrained to a feasible set V. A simple way of maintaining our updated $\mathbf{x}_t$ such that they are contained in V is to project them back to the set if they leave it.

Definition 3.3.1 (Projection)**.** Let $V \subset \mathbb{R}^n$, $V \neq \emptyset$, be a closed convex subset. The projection of an arbitrary point $\mathbf{u} \in \mathbb{R}^n$ to the subset V is defined as:

$$\Pi_V(\mathbf{u}) = \arg\min_{\mathbf{v}\in V} \|\mathbf{v} - \mathbf{u}\|_2. \quad (3.53)$$

The algorithm for OGD is as follows:

Algorithm 5 Online Gradient Descent

Input: initial value $\mathbf{x}_1 \in V$.

Require: $V \subseteq \mathbb{R}^n$ closed, convex, and non-empty.

```
1: for $t = 1, 2, \ldots, T$ do
2:     Receive loss function $\ell_t : \mathbb{R}^n \to (-\infty, +\infty]$
3:     Pay $\ell_t(\mathbf{x}_t)$
4:     $\mathbf{g}_t \leftarrow \nabla \ell_t(\mathbf{x}_t)$
5:     choose step-size $\eta_t$
6:     $\mathbf{x}_{t+1} \leftarrow \Pi_V(\mathbf{x}_t - \eta_t \mathbf{g}_t)$  $(= \arg\min_{\mathbf{v} \in V} \|\mathbf{v} - \mathbf{x}_t + \eta_t \mathbf{g}_t\|_2)$
7: end for
```

It is evident that OGD is similar to SGD in that each iteration updates via one sample. The only significant difference is that the general OGD can have different loss functions depending on the iteration. As for the projected step in line (6), this is not specific to OGD. If we choose to restrain our updates $\mathbf{x}_t$ in vanilla GD or SGD, then we can also apply a projection at the update steps. This is known in the field as projected (stochastic) gradient descent. Of course, if we don't have any constraint set V, then we simply drop the projection operation and are left with the update we've seen previously.

In terms of regret guarantees while using OGD, we have the following result:

Theorem 3.3.1. *Let $V \in \mathbb{R}^n$ be a non-empty, closed convex set that has diameter D, i.e.*

$$\max_{\mathbf{x},\mathbf{y} \in V} \|\mathbf{x} - \mathbf{y}\|_2 \leq D.$$

Let $\ell_t : V \to \mathbb{R}$ be convex, and differentiable in open sets which contain V, $t = 1, \ldots, T$. Pick any $\mathbf{x}_1 \in V$ and assume that $\eta_{t+1} \leq \eta_t$, $t = 1, \ldots, T$. Then for all $\mathbf{u} \in V$ the following regret bound holds:

$$\sum_{t=1}^{T} (\ell_t(\mathbf{x}_t) - \ell_t(\mathbf{u})) \leq \frac{D^2}{2\eta_T} + \sum_{t=1}^{T} \frac{\eta_t}{2} \|\mathbf{g}_t\|_2^2. \tag{3.54}$$

Moreover, if $\eta_t \equiv \eta$, $\forall t = 1, \ldots, T$, i.e. η_t is constant, we have:

$$\sum_{t=1}^{T} (\ell_t(\mathbf{x}_t) - \ell_t(\mathbf{u})) \leq \frac{\|\mathbf{u} - \mathbf{x}_1\|_2^2}{2\eta} + \frac{\eta}{2} \sum_{t=1}^{T} \|\mathbf{g}_t\|_2^2. \tag{3.55}$$

Proof. See [46]. □

An important observation to make is that of the step size η_t. As was mentioned earlier, this is an adversarial game, and therefore we cannot simply find the η that would minimize (3.55) (say, for the constant step size case). That is, if we do decide to choose the η that minimizes (3.55), we get that:

$$\eta^* = \frac{\|\mathbf{x}_1 - \mathbf{u}\|_2}{\sqrt{\sum_{t=1}^T \|\mathbf{g}_t\|_2^2}} \tag{3.56}$$

which would yield the regret bound:

$$\|\mathbf{x}_1 - \mathbf{u}\|_2 \sqrt{\sum_{t=1}^T \|\mathbf{g}_t\|_2^2} \tag{3.57}$$

but this requires knowledge of future gradient values, and knowing the distance between the initial value, $\mathbf{x}_1$ and $\mathbf{u}$. So the adversary can simply change the sequence of $\mathbf{y}_t$'s such that this step size no longer holds. It possible, instead, to choose η that can give us sub-linear regret if we impose that the norm of loss functions ℓ_t are bounded above by L, i.e. $\|\mathbf{g}_t\|_2 \leq L$ (or in other words, ℓ_t's are L-Lipschitz continuous by Theorem (3.2.3)), and maintaining the assumption of bounded diameter, we get:

$$\eta^* = \arg\min_{\eta} \frac{D^2}{2\eta} + \frac{\eta L^2 T}{2} = \frac{D}{L\sqrt{T}}. \tag{3.58}$$

This gives the regret bound:

$$DL\sqrt{T} \tag{3.59}$$

which is a sub-linear regret over time. As we will see, the adaptive methods apply adaptive step sizes such that the regret bounds are closer to that of the optimal regret bound (3.57).

Having the differentiability assumption on the ℓ_t's is a strong assumption and often enough, we deal with loss functions that are convex, but not differentiable. As we saw in Section (3.2.4), we can replace the gradient calculation in Algorithm (5) with a sub-gradient $\mathbf{g}_t \in \partial \ell_t(\mathbf{x}_t)$. In fact, up to changing the assumptions from differentiability to sub-differentiability, and using subgradients instead of gradients, the regret bounds in Theorem (3.3.1) also hold for online subgradient descent [46].

3.4 Radial Basis Function Interpolation

In this section we briefly touch upon radial basis function (RBF) interpolation as it is used as the basis for our results found in chapter 6 section 6.2. In general, interpolation is a general method used to approximate an unknown function f in which the only data we have on it is a finite number of pairs $(\zeta, f(\zeta))$, where $f(\zeta) \in \mathbb{R}$ is an explicit function value of f at the point $\zeta \in \mathbb{R}^n$. That is, we'd like to formulate an approximation $s : \mathbb{R}^n \to \mathbb{R}$ of f. These ideas are extendable to the more general case of approximating an unknown function $f : \mathbb{R}^n \to \mathbb{R}^m$ by approximating each component independently [11].

Let S be the linear space of approximating functions, and let $\Theta \subseteq \mathbb{R}^n$ be the set of points, ζ, such that we know the corresponding explicit function value $f(\zeta)$. Interpolation requires that we find approximating functions $s \in S$ such that they match up with f at the points $\zeta \in \Theta$, that is, for $\zeta \in \Theta$, $s(\zeta) = f(\zeta)$. In other words, we'd like to find an approximating function, s, such the function values computed by s are equal to f when computed at the points $\zeta \in \Theta$, and then with this information, interpolate unknown function values of the points between the known ζ.

In the special case of RBF interpolation, the approximating function s of f is usually a linear combination of translated real-valued functions known as radial basis functions, $\psi(\|\cdot\|)$, where $\|\cdot\|$ is the ℓ_2 norm. RBF's are such that the value of their output depends only on the ℓ_2 distance of the input to the origin, or to a translated point $\zeta \in \Theta$ which are called the centres, i.e. $\psi(\|\cdot - \zeta\|)$. Since the centres are from Θ, this creates a dependence between S on Θ. Asides from certain geometry of Θ, this dependence is important in order to avoid constructing a singular problem [11]. The term basis in RBF comes from the fact that a collection of these function $\{\psi_k\}$ are used to form a basis for a function space.

As was mentioned, the general form for the $s \in S$ are linear combinations of translated RBFs. This general form can be written explicitly as

$$s(\mathbf{x}) = \sum_{\zeta \in \Theta} \lambda_\zeta \cdot \psi(\|\mathbf{x} - \zeta\|) \tag{3.60}$$

for $\mathbf{x} \in \mathbb{R}^n$, real coefficients $\lambda_\zeta \in \mathbb{R}$, and ψ a RBF. Some well known choices of RBFs are:

$$\begin{aligned} \textbf{Gaussian:} \quad & \psi(r) = \exp -(\alpha r)^2; \\ \textbf{Multiquadrics:} \quad & \psi(r) = \sqrt{1 + (\alpha r)^2}; \\ \text{and } \textbf{Thin-Plate Splines:} \quad & \psi(r) = r \cdot \log r \end{aligned}$$

where α is a shape tuning parameter. For example, The Gaussian RBF looks like the classic bell-shaped curve, and different choices of α determine the width of the bell.

Coming back to the general form of $s(\mathbf{x})$ in (3.60), it is worth noting that this form strikes a similar resemblance to a feed-forward neural network. We can think of the RBF as being synonymous with the activation function discussed in section 3.1. In fact, this interpretation is known as a radial basis function network (RBFN) which was first introduced by Broomhead and Lowe in 1988 [9]. RBFNs usually have three layers, the input, the hidden layer, and the output layer. The hidden layer has a RBF activation function and the neurons of that layer can be thought of as performing a comparison between the input vector and the centres $\zeta \in \Theta$. The output layer is a linear combination of the RBF of each neuron in the hidden layer.

Again, referring to the general form (3.60), we see that the function s is parameterized by weights λ; in other words, we may rewrite $s(\mathbf{x})$ as $s(\mathbf{x}; \lambda)$ to show dependence on these weights. Ideally, there exists optimal weights, λ^*, such that we satisfy $s(\zeta; \lambda^*) = f(\zeta)$ for all $\zeta \in \Theta$ (finite). In order to compute the optimal weights boils down to solving a least squares problem. to see this, recall that we are equipped with a finite, say $N \in \mathbb{N}$, number of data points of the form:

$$\mathcal{D} := \{(\zeta_i, f(\zeta_i)) \; : \; \zeta_i \in \mathbb{R}^n, \; f(\zeta_i) \in \mathbb{R}, \; \forall i = 1, \ldots, N\}.$$

Using these ζ_i's, we can form radial basis functions in which their centres are the ζ_i's, i.e. $\psi(\|\mathbf{x} - \zeta_i\|)$ for $i = 1, \ldots, N$. So we want to find the weights λ_i, $i = 1, \ldots, N$ such that

$$s(\zeta_i) = \sum_{k=1}^{N} \lambda_k \cdot \psi(\|\zeta_i - \zeta_k\|) = f(\zeta_i) \tag{3.61}$$

for all $i = 1, \ldots, N$. This can be written in the form $A\lambda = \mathbf{b}$, where the entries of A are expressed as:

$$a_{ij} = \psi(\|\zeta_j - \zeta_i\|) \;\; i, j = 1, \ldots, N \tag{3.62}$$

and the solution vector is:

$$\mathbf{b}_i = f(\zeta_i) \;\; i = 1, \ldots, N. \tag{3.63}$$

As was mentioned above, uniqueness of the ζ_i is a necessary condition for the matrix A to be invertible.

3.5 ℓ_p Norm Minimization of Linear Regression

To close off the background, we briefly talk about a form of minimization problem that will be often used as comparison in the experiments of this thesis. In this section (and the in the majority of the thesis) we will be dealing with underdetermined linear regression (LR). That is, answering the problem of finding $\mathbf{x}$ such that $A\mathbf{x} = \mathbf{b}$ when $A \in \mathbb{R}^{n\times d}$ is such that $n < d$. In such a scenario, there exists an infinite number of solutions. From this space of solutions, a follow up question to ask is: which vector in this space has the smallest ℓ_p norm for $p \in [0, \infty]$?

This optimization problem can be be explicitly written as follows: let $\mathbf{A} \in \mathbb{R}^{n\times d}$ and $\mathbf{b} \in \mathbb{R}^n$. Out of the set of solutions $\mathbf{x} \in \mathbb{R}^d$ satisfying $A\mathbf{x} = \mathbf{b}$, we wish to find the the $\mathbf{x}$ with the minimum ℓ_p norm, i.e.:

$$\begin{aligned} &\min_{\mathbf{x}} \|\mathbf{x}\|_p \\ s.t \quad &A\mathbf{x} = \mathbf{b}. \end{aligned} \tag{3.64}$$

The solution to such a problem is denoted as the minimum p-norm solution where $p \in [1, \infty]$. In this thesis, we make use of ℓ_p when $p = 1, 2,$ and ∞. Given a vector $\mathbf{x} = (x_1, \dots x_n) \in \mathbb{R}^n$, the ℓ_1 norm is:

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|,$$

the ℓ_2 norm is:

$$\|\mathbf{x}\|_2 = \left(\sum_{i=1}^{n} x_i^2\right)^{1/2},$$

and the ℓ_∞ norm is:

$$\|\mathbf{x}\|_\infty = \max\{|x_1|, \dots, |x_n|\}.$$

When $p = 2$ in (3.64), we get the well known least squares problem.

As was mentioned, ℓ_p-norm minimization of overparameterized linear regression will be considered when we characterize the convergence behaviour of the adaptive method Adagrad [18]. Namely, we use the solutions outputted by the minimum p-norm LR ($p \in$

$\{1, 2, \infty\}$) as benchmarks to compare and designate how diffuse the solutions outputted by (stochastic) Adagrad are. We'll see that, under a metric to compute how diffuse a solution is, (stochastic) Adagrad promotes diffuse solutions like the minimum ∞-norm LR solution (see Chapter 6 Section 6.1).

Chapter 4

Adaptive Gradient Methods

In this section, we dive into explaining three highly used adaptive gradient methods. Before that, however, we briefly discuss momentum as it helps us in the discussion of the adpative methods, mainly RMSProp, and Adam.

4.1 Momentum

In Section (3.2.3), we discussed mini-batch GD. In short, it is a trade off between between SGD and GD by running each iteration on a partition (larger than 1 and less than the total number of samples) of the data. By updating our weights in this fashion, we end up with a zigzag motion towards the solution, see Figure (4.1)**(a)**. If we refer to the direction pointing to the minimum as "down", then it is evident from the zigzagging pattern that our steps are spending too much time moving left and right. The idea behind momentum is to modify the update in a way such that we reduce movement in the left and right directions, and promote movement in the downward direction. That is, we are attempting to smooth out the zigzag such that the majority of the step is spent in the downward direction.

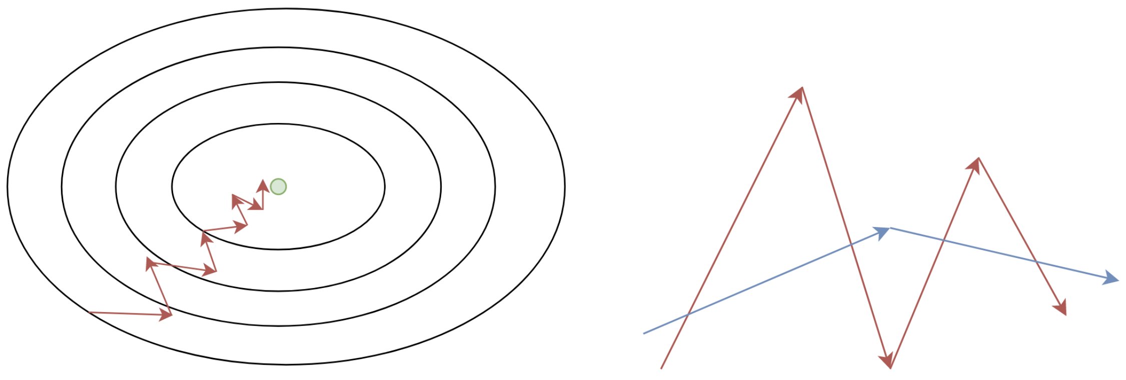


Figure 4.1: **(a)**: Arbitrary contour plot with optimal minimum in green. Red arrows represent the direction of descent of mini-batch gradient descent. **(b)** red arrows: normal GD update, blue arrows: momentum adjusted updates.

In order to represent this smoothing idea mathematically requires knowledge of the *exponentially weighted moving average* (EWMA) (also referred to as *exponentially weighted average*). Given data over a time frame (think time series data), EWMA is a method to calculate the trend of the data over time using the average of local data points, or, in other words, a moving average of the data. Let θ_t be a data point at time t, and let V_t be the EWMA of the data at time t. Then we have that:

$$\begin{aligned} V_0 &= 0 \\ V_t &= \beta V_{t-1} + (1-\beta)\theta_t \quad t > 0 \end{aligned} \tag{4.1}$$

where $\beta \in (0,1)$ is a hyperparameter. The choice of β tells us how far back in time we wish to use data in the computation of the current EWMA. It is important to note that as time passes, the weight given in the average to farther points θ_t are exponentially less pronounced than closer data points in time. To see this exponential decay more clearly, one can expand out (4.1), to get:

$$V_t = (1-\beta)\sum_{i=1}^{t} \beta^{t-i}\theta_i \tag{4.2}$$

so we see that, as we move further away from the current time t, the weight associated with data point θ_i in the average decays exponentially by $(1-\beta)\beta^i$.

We are now in a position to introduce the momentum algorithm. Recall the weights update for gradient descent:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \nabla f(\mathbf{x}_t)$$

what momentum does, is replace the gradient update with the exponentially weighted moving average of the past gradients. That is, if we define $V_{\partial \mathbf{x}_t}$ to be the EWMA of $\nabla f(\mathbf{x}_t)$ at time t, we get the following update rule:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta V_{\partial \mathbf{x}_t} \tag{4.3}$$

which is the momentum algorithm's update rule. The full algorithm applied to gradient descent is as follows:

Algorithm 6 Gradient Descent with Momentum

Input: initial value $\mathbf{x}_0 \in \mathbb{R}^d$, continuously differentiable function $f : \mathbb{R}^d \to \mathbb{R}$, EWMA hyperparameter $\beta \in (0, 1)$
Output: optimal solution $\mathbf{x}^* \in \mathbb{R}^d$

```
1: $V_{\partial \mathbf{x}_0} \leftarrow 0$
2: $\mathbf{x}_1 \leftarrow \mathbf{x}_0$
3: for $t = 1, 2, 3 \dots$ do
4:     $\mathbf{g}_t \leftarrow \nabla f(\mathbf{x}_t)$
5:     $V_{\partial \mathbf{x}_t} \leftarrow \beta V_{\partial \mathbf{x}_{t-1}} + (1 - \beta)\mathbf{g}_t$
6:     choose step-size $\eta_t$
7:     $\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t - \eta_t V_{\partial \mathbf{x}_t}$
8: end for
```

simply changing line (4) in Algorithm (6) to mini-batch gradients converts the algorithm to mini-batch GD with momentum.

To intuitively see why this promotes a less oscillatory "downward" direction we spoke of above, notice that when we take the averages of the steps, the left and right directions that promote the zigzag behaviour will average to approximately 0 since they roughly point in opposite directions, while every step always moves in the downward direction so that when taking the average among the steps, we continue to point down, see Figure (4.1)**(b)**.

4.2 Adagrad

The theme of large data has been a staple throughout the text so far. With this setting comes data sets with large dimensional feature spaces, many features of which are irrelevant. However, it is common to have feature spaces that contain certain, more rare, features that are information rich. Under the gradient-based methods discussed so far, we apply the same learning rate across all features. Additionally, we briefly spoke of momentum as a method to promote a better direction of descent by updating the weights equally in terms of exponentially weighted moving averages. However, doing so, the optimizer fails to promote these more sparse, yet highly informative, features compared to other less-sparse features. Adagrad [18], short for **Ada**ptive **Grad**ient, aims to correct this shortcoming by dynamically applying feature specific learning rates catered towards each feature based on the geometry of past gradient information. Informally, Adagrad will give frequently seen features a lower learning rate, while giving infrequent features a higher learning rate [18].

Adagrad was originally written in the online learning setting which we briefly discussed in section (3.3). However, as was mentioned, there is a connection between the online learning setting and the stochastic setting which will allow us to easily connect the regret bound to a convergence guarentee for the stochastic convex setting. For now, we dive into the construction of Adagrad and its results in the online setting as it was originally.

Recall the online learning setting. In this setting, one wishes to minimize the regret with respect to a static predictor $\mathbf{u}^* \in V$ where $V \subseteq \mathbb{R}^n$ is a closed convex set and $\mathbf{u}^* = \arg\min_{\mathbf{u}\in V} \sum_{t=1}^{T} f_t(\mathbf{u})$. Specifically, given a sequence of functions f_t, one aims to get the smallest possible solution to:

$$R(T) = \sum_{t=1}^{T} f_t(\mathbf{x}_t) - \sum_{t=1}^{T} f_t(\mathbf{u}^*). \tag{4.4}$$

From here, at each iteration t, we are given a subgradient $\mathbf{g}_t = \partial f_t(\mathbf{x}_t)$, and one can use stochastic projected gradient update (see Algorithm (5)) to get

$$\mathbf{x}_{t+1} = \Pi_V(\mathbf{x}_{t+1}) = \arg\min_{\mathbf{u}\in V} \|\mathbf{u} - \mathbf{x}_t + \eta \mathbf{g}_t\|_2^2 \tag{4.5}$$

and we'd get the same analysis as we saw in section (3.3). Namely, we'd get the regret bound (3.55) which we rewrite here with respect to f_t:

$$\sum_{t=1}^{T} (f_t(\mathbf{x}_t) - f_t(\mathbf{u})) \le \frac{1}{2\eta}\|\mathbf{u} - \mathbf{x}_1\|_2^2 + \frac{\eta}{2}\sum_{t=1}^{T} \|\mathbf{g}_t\|_2^2. \tag{4.6}$$

Duchi et al, [18] proposed to change the the L_2 norm of the projection to the Mahalanobis norm. Given a matrix $A \succeq 0$, the Mahalanobis norm is defined to be

$$\|\cdot\|_A = \sqrt{\langle \cdot, A\cdot \rangle}. \tag{4.7}$$

Using this norm, we'll define the projection onto a set V using the Mahalanobis norm with respect to the matrix A as:

$$\Pi_V^A(\mathbf{x}) = \arg\min_{\mathbf{v}\in V} \|\mathbf{v} - \mathbf{x}\|_A^2 = \arg\min_{\mathbf{v}\in V} \langle (\mathbf{v} - \mathbf{x}), A(\mathbf{v} - \mathbf{x})\rangle. \tag{4.8}$$

The new generalized update to (4.5) is then written as:

$$\mathbf{x}_{t+1} = \Pi_V^A(\mathbf{x}_t - \eta A^{-1}\mathbf{g}_t) = \arg\min_{\mathbf{v}\in V} \|\mathbf{v} - (\mathbf{x}_t - \eta A^{-1}\mathbf{g}_t)\|_A^2. \tag{4.9}$$

Under this generalized update, the standard regret (4.6) has the form:

$$\sum_{t=1}^{T}(f_t(\mathbf{x}_t) - f_t(\mathbf{u})) \leq \frac{1}{2\eta}\|\mathbf{u} - \mathbf{x}_1\|_A^2 + \frac{\eta}{2}\sum_{t=1}^{T}\|\mathbf{g}_t\|_{A^{-1}}^2. \tag{4.10}$$

From here, we wish to minimize the regret upper bound with respect to A. That is, we want to find the optimal value of the following minimization problem:

$$\begin{aligned} \min_A & \sum_{t=1}^{T}\langle \mathbf{g}_t, A^{-1}\mathbf{g}_t\rangle \\ \text{s.t } & A \succeq 0 \\ & tr(A) \leq C \end{aligned} \tag{4.11}$$

where $tr(\cdot)$ is the trace of the input. The A that minimizes (4.11) is proven to be [18]:

$$A = c\left(\sum_{t=1}^{T}\mathbf{g}_t\mathbf{g}_t^\top\right)^{1/2} \tag{4.12}$$

where c is chosen to satisfy $tr(A) \leq C$ from (4.11). The authors derive this optimal solution in the proof of [18, Appendix E, Lemma 15] by considering two cases, where in each case they utilize the Lagrangian of (4.11). The first case our matrix A is full rank and therefore positive definite; This allows the authors to directly solve for the solution. In the second case, they consider A not full rank (hence positive semi-definite by the first constraint of (4.11)). In this case, the authors work with the dual and show that strong duality holds and that the limiting solution which closes the gap is of the same form as the first case, except under the pseudo-inverse.

Under the generalized update (4.9) and choosing A to satisfy (4.12), we get the following Adagrad algorithm for online learning:

Algorithm 7 Online Adagrad

Input: Step size $\eta > 0$, perturbation parameter $\epsilon \geq 0$
Initialize: $\mathbf{x}_1 = \mathbf{0}$, $G_0 = \mathbf{0}_{d\times d}$, $H_0 = \mathbf{0}_{d\times d}$
Output: optimal solution $\mathbf{x}^* \in V$

```
1: for $t = 1, 2, \dots, T$ do
2:     Receive loss function $f_t : \mathbb{R}^n \to (-\infty, +\infty]$
3:     Pay $f_t(\mathbf{x}_t)$
4:     $\mathbf{g}_t \leftarrow \partial f_t(\mathbf{x}_t)$
5:     $G_t = G_{t-1} + \mathbf{g}_t\mathbf{g}_t^\top$
6:     $H_t = (G_t + \epsilon I)^{1/2}$
7:     $\mathbf{x}_{t+1} \leftarrow \Pi_V^{H_t}(\mathbf{x}_t - \eta H_t^{-1}\mathbf{g}_t)$  $(= \arg\min_{\mathbf{v}\in V} \|\mathbf{v} - \mathbf{x}_t + \eta H_t^{-1}\mathbf{g}_t\|_{H_t}^2)$
8: end for
```

In practice, however, Algorithm (7) requires one to compute the root and inverse of a matrix at each iteration t which is not practical to compute when dealing with a high number of dimensions. To overcome this, we simply use the diagonal of H_t at each iteration [18], that is, we change line (7) to

$$\mathbf{x}_{t+1} \leftarrow \Pi_V^{\mathrm{Diag}(H_t)}(\mathbf{x}_t - \eta \mathrm{Diag}(H_t)^{-1}\mathbf{g}_t) \tag{4.13}$$

where Diag$(\cdot)$ returns the the input with its diagonal elements unchanged, and the rest are set to 0.

With (both full and diagonal) Adagrad constructed, we now turn to their theoretical regret guarantees. Let

$$R_2 := \max_{t\leq T} \|\mathbf{x}_t - \mathbf{x}^*\|_2 \tag{4.14}$$

$$\text{and, } R_\infty := \max_{t\leq T} \|\mathbf{x}_t - \mathbf{x}^*\|_\infty \tag{4.15}$$

where $\mathbf{x}^* = \inf_{\mathbf{x}\in V} \sum_{t=1}^T f_t(\mathbf{x})$. Furthermore, we denote $\mathbf{g}_{1:t} = [\mathbf{g}_1, \mathbf{g}_2, \dots, \mathbf{g}_t]$, to be the matrix of concatenated subgradients from the sequence up to time t. Finally, we denote the j-th row of the matrix $\mathbf{g}_{1:t}$ as $\mathbf{g}_{1:t,j}$, which can be viewed as the j-th coordinate of each subgradient in the sequence up to time t. Then we have the following regret bound guarantees:

Theorem 4.2.1. *Let $\{\mathbf{x}_t\}_{t=1}^T$ be the sequence generated by Algorithm (7) for full-matrix H_t. Then Adagrad suffers the following regret bound:*

$$\sum_{t=1}^{T} f_t(\mathbf{x}_t) - f_t(\mathbf{x}^*) \leq R_2 \cdot tr(H_T). \tag{4.16}$$

Furthermore, if we consider the sequence $\{\mathbf{x}_t\}_{t=1}^T$ generated by Algorithm (7) but with update (4.13), i.e. diagonal matrix H_t, then Adagrad suffers the following regret bound:

$$\sum_{t=1}^{T} f_t(\mathbf{x}_t) - f_t(\mathbf{x}^*) \leq 2R_\infty \sum_{j=1}^{d} \|\mathbf{g}_{1:T,j}\|_2. \tag{4.17}$$

Proof. See [18]. □

A consequence of these regret bounds is that we may use them to formulate convergence guarantees for Adagrad in the stochastic convex optimization setting. Recall that in the stochastic setting, we aimed at minimizing the expected risk (3.34). For simplicity we'll define $f(\mathbf{x})$ to be the empirical risk, i.e. $f(\mathbf{x}) := \mathbb{E}[h(\mathbf{x};\zeta)]$, and define $f_t(\mathbf{x}) := h(\mathbf{x};\zeta_t)$. Then we have the following convergence guarantee for diagonal approximated Adagrad in the stochastic convex setting [19]:

$$\mathbb{E}\left[f\left(\frac{1}{T}\sum_{t=1}^{T}\mathbf{x}_t\right)\right] - f(\mathbf{x}^*) \leq \frac{2R_\infty}{T}\sum_{i=1}^{d}\mathbb{E}\left[\|\mathbf{g}_{1:T,j}\|_2\right] \tag{4.18}$$

where d is the number of dimensions. **Unless specified otherwise, when we speak of Adagrad, moving forward, we speak of the diagonal approximated version of Adagrad.**

One disadvantage to using Adagrad in the "offline" setting is that over time, the diagonal entries of H_t^{-1} will shrink to zero. This causes Adagrad to converge more more slowly. To see why they shrink to zero, lets focus on the unconstrained Adagrad update:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \mathrm{Diag}(H_t)^{-1}\mathbf{g}_t. \tag{4.19}$$

If we inspect the update at an arbitrary coordinate of $\mathbf{x}_{t+1}$, say:

$$(\mathbf{x}_i)_{t+1} = (\mathbf{x}_i)_t - \eta \frac{1}{\sqrt{\sum_{\tau=1}^{t} (\mathbf{g}_\tau)_i^2 + \epsilon}} (\mathbf{g}_\tau)_i \tag{4.20}$$

for some $i \in \{1, \dots, d\}$, then we see that as $t \to \infty$, the denominator grows to infinity under the assumption that the $(\mathbf{g}_\tau)_i \neq 0$. Taking the reciprocal, the learning rate goes to 0. This disadvantage is overcome in both RMSProp and Adam, both which are discussed in this chapter.

4.3 RMSProp and Adadelta

It was briefly spoken about that Adagrad suffers slow training times due to the accumulation of all past gradients in its preconditioner H_t. *RMSProp*, short for **R**oot **M**ean **S**quared **Prop**agation is an unpublished adaptive method proposed by Geoffrey Hinton in his Coursera lecture titled "Neural Networks for Deep Learning", lecture 6e [29]. This method can be seen as an extension of Adagrad with the ability to overcome the slow training times commonly present with Adagrad. At a high level, RMSProp follows the same style of update rule as Adagrad but opts to accumulate past squared gradients over a window instead of considering all previous squared gradients. Using a window of past square gradients prevents H_t^{-1} in Adagrad to possibly go to infinity.

To be more concrete, RMSProp creates this "window" by utilising the exponentially weighted moving average of the previous squared gradients $\mathbf{g}_t^2$ (here the squared is applied entry-wise via Hadamard product). The choice of using an EWMA is used to overcome the inefficiencies of having to store an actual range of past squared gradients [63]. Let us denote the EWMA of the squared gradient at time t to be $\mathbb{E}[\mathbf{g}^2]_t$. Then, by equation (4.1), we have that

$$\mathbb{E}[\mathbf{g}^2]_t = \gamma \mathbb{E}[\mathbf{g}^2]_{t-1} + (1-\gamma)\mathbf{g}_t^2. \tag{4.21}$$

Now recall the Adagrad update step, namely,

$$\Delta \mathbf{x}_t = -\frac{\eta}{\sqrt{G_t^{1/2} + \epsilon I}} \cdot \mathbf{g}_t \tag{4.22}$$

where $\Delta \mathbf{x}_t = \mathbf{x}_{t+1} - \mathbf{x}_t$, and the inverse square root operations on G_t and $G_t^{1/2} + \epsilon I$ are applied entry-wise. RMSProp then exchanges $G_t^{1/2}$ with the EWMA (4.21) to get the new update rule:

$$\Delta \mathbf{x}_t = -\frac{\eta}{\sqrt{\mathbb{E}[\mathbf{g}^2]_t + \epsilon I}} \cdot \mathbf{g}_t. \tag{4.23}$$

We note that the denominator term $\sqrt{\mathbb{E}[\mathbf{g}^2]_t}$ is exactly the root of the mean of squared gradients and hence where the name comes from. Letting $RMS(\mathbf{g}_t) := \sqrt{\mathbb{E}[\mathbf{g}^2]_t + \epsilon I}$ we may rewrite (4.23) as

$$\Delta \mathbf{x}_t = -\frac{\eta}{RMS(\mathbf{g}_t)} \cdot \mathbf{g}_t. \tag{4.24}$$

Around the same time, another adaptive method was developed (independent of RMSProp) by Zeiler known as Adadelta [63]. This method derived the same update rule (4.23) as RMSProp. The key difference then was in the additional observation made by the author in which he noticed that, under the learning rates present in SGD, momentum, Adagrad, or (4.23), the learning rates do not share the same "units" as that of the parameter $\mathbf{x}$. Using Adagrad as an example, the adaptive learning rates are of units proportional to one over the unit of the gradient. As such, the update, which is a ratio of gradients is unitless.

To ensure consistency of the units when updating the parameter, the author utilises the fact that the inverse Hessian (or an approximation of) do have the correct units; if $\mathbf{H}$ denotes the Hessian matrix then [63]:

$$\Delta \mathbf{x} \propto \mathbf{H}^{-1}\mathbf{g} \propto \frac{\frac{\partial f}{\partial \mathbf{x}}}{\frac{\partial^2 f}{\partial \mathbf{x}^2}} \propto \text{units of } \mathbf{x}. \tag{4.25}$$

Using this information, we can come up with additional terms to add to (4.23) such that the units match that of the parameter $\mathbf{x}$. Specifically, under the assumption that the Hessian is diagonal, the author utilises Newton's method to get

$$\Delta \mathbf{x} = \frac{\frac{\partial f}{\partial \mathbf{x}}}{\frac{\partial^2 f}{\partial \mathbf{x}^2}} \implies \frac{1}{\frac{\partial^2 f}{\partial \mathbf{x}^2}} = \frac{\Delta \mathbf{x}}{\frac{\partial f}{\partial \mathbf{x}}}. \tag{4.26}$$

As there is already units of the gradient in the denominator thanks to $RMS(\mathbf{g}_t)$, we just need to include $\Delta \mathbf{x}$ in the numerator to get the update rule:

$$\Delta \mathbf{x}_t = -\frac{\Delta \mathbf{x}_t}{RMS(\mathbf{g}_t)} \cdot \mathbf{g}_t. \tag{4.27}$$

Of course, $\Delta \mathbf{x}_t$ is not known at time t, so we must add the extra assumption that the curvature is locally smooth to then approximate $\Delta \mathbf{x}_t$ by using $RMS(\Delta \mathbf{x}_{t-1})$. Specifically, we consider the EWMA of $\Delta \mathbf{x}_t^2$ by

$$\mathbb{E}[\Delta \mathbf{x}^2]_t = \gamma \mathbb{E}[\Delta \mathbf{x}^2]_{t-1} + (1-\gamma)\Delta \mathbf{x}_t^2. \tag{4.28}$$

and then writing the update rule to be

$$\Delta \mathbf{x}_t = -\frac{RMS(\Delta \mathbf{x}_{t-1})}{RMS(\mathbf{g}_t)} \cdot \mathbf{g}_t. \tag{4.29}$$

Note that the step size η was replaced with $RMS(\Delta \mathbf{x}_{t-1})$ in that Adadelta does not have a tuneable step size parameter. Since this is the case, it is important that $RMS(\Delta \mathbf{x}_{t-1})$ contains the same ϵ perturbation parameter present in $RMS(\mathbf{g}_t)$ to allow the start off of the first iteration, and to allow for progress in the event where the parameters end up being small. Additionally, both Hinton and Zeiler recommend setting $\gamma = 0.9$ in practice for the EWMA γ parameter (for both RMSProp and Adadelta).

4.4 Adam

The theme we've seen in chapter (4) so far is that each succeeding section builds off the previous section(s). This theme continues with the introduction to the Adam optimizer. Adam, short for **Ada**ptive **m**oment estimation, is the final adaptive gradient method we'll introduce in this thesis. Proposed by Kingma and Ba [36], Adam aims to combine the advantages present in Adagrad in which good training performance remains under the event of sparse gradients, and RMSProp which has seen strong success in online learning (and thus stochastic optimization). It is also common to compare Adam to RMSProp with momentum, but it has certain tweaks and construction differences that give it stronger advantages.

Like RMSProp, Adam utilises the EWMA of past squared gradients. Additionally, it also utilises the EWMA of past gradients. We can think of these as estimates of the second (raw) and first moments of the gradient (uncentered variance, and mean), respectively. If we let m_t denote the EWMA of the first moment of $\mathbf{g}_t$, and v_t as the EWMA of the second moment of $\mathbf{g}_t$, we have

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)\mathbf{g}_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)\mathbf{g}_t^2. \end{aligned} \tag{4.30}$$

where the authors' recommend setting $\beta_1 = 0.9$, and $\beta_2 = 0.999$. Using these, we get the update rule:

$$\Delta x_t = -\frac{\eta}{\sqrt{v_t + \epsilon}} \cdot m_t. \tag{4.31}$$

One issue with this update rule is that, since they are initialized to zero, i.e. $m_0 = v_0 = \mathbf{0}$, these estimates of the first moment and second (raw) moment are biased towards zero, especially at the early iterations of the algorithm [36]. It is possible to overcome this bias by considering the following analysis for the second (raw) moment (the analysis for the first moment is similar). Given a stochastic objective function f, let the sequence of gradients of f up to time step T each be drawn from some distribution $P(\mathbf{g}_t)$, i.e. $\mathbf{g}_t \sim P(\mathbf{g}_t)$. Next, recall the EWMA update for the squared gradient:

$$\begin{aligned} v_0 &= \mathbf{0} \\ v_t &= \beta_1 v_{t-1} + (1-\beta_1)\mathbf{g}_t^2 \end{aligned} \tag{4.32}$$

our goal is to see if we can find some relation between $\mathbb{E}[v]_t$ and $\mathbb{E}[\mathbf{g}^2]_t$ such that we can derive a "bias correction" term, say $v_{fix}^{(t)}$, such that $v_{fix}^{(t)} \cdot \mathbb{E}[v]_t = \mathbb{E}[\mathbf{g}^2]_t$. Recalling that we may write the EWMA as in equation (4.2), we have that [36]:

$$\mathbb{E}[v_t] = \mathbb{E}\left[(1-\beta_2)\sum_{i=1}^{t} \beta_2^{t-i}\mathbf{g}_i^2\right] \tag{4.33}$$

$$= \mathbb{E}[\mathbf{g}^2]_t \cdot (1-\beta_2)\sum_{i=1}^{t} \beta_2^{t-1} + \zeta \tag{4.34}$$

$$= \mathbb{E}[\mathbf{g}^2]_t \cdot (1-\beta_2^t) + \zeta. \tag{4.35}$$

Here, ζ is kept small due to choosing β_1 such that the weight assigned to gradients far in the past are small, or $\zeta = 0$ if the $\mathbb{E}[\mathbf{g}^2]_i$ are stationary (i.e. mean and variance are constant over time). So we see that if we choose $v_{fix}^{(t)} = 1/(1-\beta_2^t)$, then we have that $v_{fix}^{(t)} \cdot \mathbb{E}[v]_t \approx \mathbb{E}[\mathbf{g}^2]_t$. Similarly, it can be derived that the bias correcting term for the first moment, $m_{fix}^{(t)} = 1/(1-\beta_1^t)$. Using these bias correcting terms, we get the official Adam update:

$$\Delta \mathbf{x}_t = -\frac{\eta}{\sqrt{\frac{v_t}{(1-\beta_2^t)} + \epsilon}} \cdot \frac{m_t}{(1-\beta_1^t)}. \tag{4.36}$$

Letting $\hat{m}_t := m_t/(1-\beta_1^t)$ and $\hat{v}_t := v_t/(1-\beta_2^t)$, we neatly rewrite (4.36) as

$$\Delta \mathbf{x}_t = -\frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \cdot \hat{m}_t. \tag{4.37}$$

This yield the following algorithm for Adam:

Algorithm 8 Adam

Input: Step size $\eta > 0$, perturbation parameter $\epsilon \geq 0$, decay β_1, $\beta_2 \in [0,1)$
Initialize: $\mathbf{x}_0 = \mathbf{0}$, $m_0 = 0$, $v_0 = 0$.
Output: optimal solution $\mathbf{x}^* \in \mathbb{R}^n$

```
1: for $t = 1, 2, \dots$ do
2:     Receive loss function $f_t : \mathbb{R}^n \to (-\infty, +\infty]$
3:     $\mathbf{g}_t \leftarrow \partial f_t(\mathbf{x}_{t-1})$
4:     $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1)\mathbf{g}_t$
5:     $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2)\mathbf{g}_t^2$
6:     $\hat{m}_t \leftarrow m_t/(1-\beta_1^t)$
7:     $\hat{v}_t \leftarrow v_t/(1-\beta_2^t)$
8:     $\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} - \eta \cdot \hat{m}_t/\sqrt{\hat{v}_t + \epsilon}$
9: end for
```

With the algorithm established, we turn to the convergence analysis of Adam. Similar to Adagrad, the authors of Adam analyse convergence of Adam using the online learning setting. Let $\mathbf{g}_t = \nabla f_t(\mathbf{x}_t)$, $\mathbf{g}_{1:t} = [\mathbf{g}_1, \dots, \mathbf{g}_t]$ where $\mathbf{g}_{1:t,j} = [\mathbf{g}_{1,j}, \dots, \mathbf{g}_{t,j}]$ is the j-th coordinate of each gradient in the sequence up to time t, the regret function $R(T)$ defined as in equation (4.4) with $\mathbf{u} = \arg\min_{\mathbf{u}\in\mathbb{R}^n} f_t(\mathbf{u})$, and $\gamma := \beta_1^2/\sqrt{\beta_2}$. Then we have the following regret bound for Adam

Theorem 4.4.1. *Assume the gradients of f_t are bounded as: $\|\nabla f_t(\mathbf{x})\|_2 \leq G$ and $\|\nabla f_t(\mathbf{x})\|_\infty \leq G_\infty$ for all $\mathbf{x} \in \mathbb{R}^d$. Assume that the distance between any $\mathbf{x}_t$ generated by Adam is bounded, $\|\mathbf{x}_j - \mathbf{x}_k\|_2 \leq D$ and $\|\mathbf{x}_j - \mathbf{x}_k\|_\infty \leq D_\infty$ for any $j, k \in \{1, \dots, T\}$. Finally, assume that $\beta_1, \beta_2 \in [0,1)$ such that they satisfy $\beta_1^2/\sqrt{\beta_2} < 1$. Let step size $\eta_t = \eta/\sqrt{t}$, and $\beta_{1,t} = \beta_1\lambda^{t-1}$, $\lambda \in (0,1)$. Adam suffers the follow regret bound, for all $T > 1$:*

$$R(T) \leq \frac{D^2}{2\eta(1-\beta_1)} \sum_{i=1}^{d} \sqrt{T\hat{v}_{T,i}} + \frac{\eta(1+\beta_1)G_\infty}{(1-\beta_1)\sqrt{1-\beta_2}(1-\gamma)^2} \sum_{i=1}^{d} \|\mathbf{g}_{1:T,i}\|_2 + \sum_{i=1}^{d} \frac{D_\infty^2 G_\infty \sqrt{1-\beta_2}}{2\eta(1-\beta_1)(1-\gamma)^2}.$$

Proof. See [36]. □

We note that in order for the theorem to hold, one requires that the learning rate η decays by a rate of $1/\sqrt{t}$ and that the EWMA decay parameter β_1 decays exponentially with λ which is represented with $\beta_{1,t}$ in the theorem. Using Theorem (4.4.1), we have the following corollary which tells us that the average regret produced by Adam converges

Corollary 4.4.1.1. *Assume the gradients of f_t are bounded as: $\|\nabla f_t(\mathbf{x})\|_2 \leq G$ and $\|\nabla f_t(\mathbf{x})\|_\infty \leq G_\infty$ for all $\mathbf{x} \in \mathbb{R}^d$. Assume that the distance between any $\mathbf{x}_t$ generated by Adam is bounded, $\|\mathbf{x}_j - \mathbf{x}_k\|_2 \leq D$ and $\|\mathbf{x}_j - \mathbf{x}_k\|_\infty \leq D_\infty$ for any $j, k \in \{1, \ldots, T\}$. Then Adam achieves the following regret guarantee:*

$$\frac{R(T)}{\sqrt{T}} \leq O\left(\frac{1}{\sqrt{T}}\right). \tag{4.38}$$

Proof. See [36]. □

We conclude this section by noting that Adam has some extensions one which can be found in the original paper [36] called AdaMax which extends proportionality of v_t from the L_2 norm to L_p norm, and another example being Nadam [17] which combines Adam with Nesterov accelerated gradient (NAG) [44], and hence the name **N**esterov-accelerated **ada**ptive **m**oment estimation.

Chapter 5

Algorithmic Behaviour of Adagrad in Underdetermined Linear Regression

5.1 Motivation and the Setting of Study

As was briefly discussed in the introduction, it is common when training deep learning models that the number of parameters/features outnumber the number of training instances. In this regime, known as over-parameterized, there will exist a large number of elements in the space of solutions. That is, there are plenty of minima one can land in when optimizing the loss function. Many of these solutions, while optimal in the training phase, do not yield parameters that allow the model to generalize well. It turns out that when using SGD to minimize, the minima that the algorithm biases towards (i.e. the one it chooses) tends to generalize well [34]. The study of the algorithmic behaviour, or in other terms its implicit bias, is thus an important factor since it can determine whether your model generalizes well or not.

We'd like to carry over this study of implicit biases for the adaptive methods. In particular, we explore the behaviour of the Adagrad algorithm. We saw in chapter (3.1) that MLPs, a basic deep learning structure, is the composition of many possibly non-linear functions (see equation (3.7)). The number of compositions depends on the number of layers the network sports, and the functions composed are heuristically chosen activation functions. Attempting to study the algorithmic behaviours of Adagrad under such a complex function is difficult. We therefore limit our study to the following linear regression

(LR) case:

$$f(\mathbf{x}) = A\mathbf{x} - \mathbf{b} \tag{5.1}$$

where $A \in \mathbb{R}^{n \times d}$ is underdetermined ($n \leq d$), and where one can think of (5.1) as a single layered perceptron with activation function equal to the identity function $\sigma(x) = x$. So, while basic in nature, studying the algorithmic behaviour under this regime, which we'll denote as the *underdetermined LR* regime, can be thought of as a very simple analog of a MLP. Furthermore, we study the implicit biases of Adagrad under this regime using the sum of squared errors loss function

$$\ell(\mathbf{x}) = \frac{1}{2}\|A\mathbf{x} - \mathbf{b}\|_2^2. \tag{5.2}$$

In all, we aim to study and discover the behaviours of Adagrad when minimizing the following unconstrained optimization problem:

$$\min_{\mathbf{x} \in \mathbf{R}^d} \frac{1}{2}\|A\mathbf{x} - \mathbf{b}\|_2^2. \tag{5.3}$$

5.2 A Simple Scenario: Fixed Preconditioned Gradient Descent

Based on our initial study of Adagrad in section (4.2), we saw that we may write the Adagrad update rule as the following:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta(G_k^{1/2})^{-1} \cdot \mathbf{g}_k \tag{5.4}$$

where $\mathbf{g}_k = \nabla f(\mathbf{x}_k)$, and the preconditioner matrix $(G_k^{1/2})^{-1} \in \mathbb{R}^{d \times d}$ is a diagonal positive definite matrix where the i-th diagonal element is written as:

$$(G_k^{1/2})_i^{-1} = \frac{1}{\sqrt{\sum_{\tau=0}^{k}(\mathbf{g}_\tau)_i^2 + k\epsilon}} \tag{5.5}$$

where $(\mathbf{g}_k)_i$ is the i-th coordinate of the gradient of f at $\mathbf{x}_k$. We note here that the update (5.5) is slightly different than the update provided in section (4.2) in that we include the perturbation parameter ϵ in the sum of the update. This was chosen due to the fact that the ϵ perturbation parameter is chosen in a data-dependent manner in order to prove the regret bounds [18, Theorem 5]. In order to satisfy this in a universal manner, we grow ϵ on each iteration so that we are assured that this dependence is eventually met.

As the preconditioner matrix changes with each iteration k, this behaviour increases the difficulty in showing convergence of Adagrad under (5.3). Instead, we initially consider an iterative method when the preconditioner matrix is diagonal, positive definite and fixed for each iteration k. Let G represent this fixed preconditioner, then we have the update rule:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta G \cdot \mathbf{g}_k \tag{5.6}$$

which can be thought of as gradient descent but with fixed weights applied to each partial derivative of the gradient independently. With respect to the underdetermined problem (5.3), theorem 5.2.1 shows us that fixed preconditioner gradient descent implicitly biases its solution towards the minimum 2-norm solution of $(AG^{1/2})\mathbf{x} = \mathbf{b}$.

Theorem 5.2.1. *Let $f(\mathbf{x})$ be as in (5.2) where $A \in \mathbb{R}^{n\times d}$ $(n < d)$ and full row rank, $\mathbf{x} \in \mathbb{R}^d$, and $\mathbf{b} \in \mathbb{R}^n$, and let $\mathbf{g}_k = \nabla f(\mathbf{x}_k)$. Let $G \in \mathbb{R}^{d\times d}$ be a fixed positive definite matrix. Consider the fixed preconditioner GD update*

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta G \cdot \mathbf{g}_k, \tag{5.7}$$

where $\mathbf{x}_0 = \mathbf{0}$. Then the sequence generated by this update, $\{\mathbf{x}_k\}_{k=0}^{\infty}$ converges to

$$\mathbf{x}^* = (AG^{1/2})^T(AG^{1/2}(AG^{1/2})^T)^{-1}b \tag{5.8}$$

Proof. We show this as follows. Let $\mathbf{y}_k = G^{-1/2}\mathbf{x}_k$. Then we can rewrite $f(\mathbf{x})$ as:

$$\frac{1}{2}\|AG^{1/2}\mathbf{y} - \mathbf{b}\|_2^2 \tag{5.9}$$

which we can think of as the least-squares problem using the underdetermined matrix $AG^{1/2}$. From here, we show that running gradient descent on (5.9) is equivalent to running fixed-preconditioner GD on $f(\mathbf{x})$. Deriving the GD update of (5.9) we get:

$$\nabla_{\mathbf{y}}\left[\frac{1}{2}\|AG^{1/2}\mathbf{y} - \mathbf{b}\|_2^2\right] = (AG^{1/2})^T(AG^{1/2}\mathbf{y} - \mathbf{b}). \tag{5.10}$$

This yields us the standard GD update:

$$\mathbf{y}_{k+1} = \mathbf{y}_k - \eta(AG^{1/2})^T(AG^{1/2}\mathbf{y}_k - \mathbf{b}). \tag{5.11}$$

Since $(AG^{1/2})$ is underdetermined, and $\mathbf{x}_0 = \mathbf{0}$, we know that this sequence of updates will converge to the minimum 2-norm solution of (5.9). Now, substituting back for $\mathbf{x}_k$, we get that this update is equivalent to:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta G^{1/2}(AG^{1/2})^T(A\mathbf{x}_k - \mathbf{b}) \tag{5.12}$$
$$= \mathbf{x}_k - \eta G A^T(A\mathbf{x}_k - \mathbf{b}) \tag{5.13}$$

which is the update for fixed-preconditioner GD. Since these updates are equivalent, we conclude that fixed-preconditioner GD applied to $f(\mathbf{x})$ converges to the minimum 2-norm solution of (5.9).
This concludes the proof. □

Theorem (5.2.1) tells us that fixed preconditioner gradient descent converges to the minimum 2-norm solution of $(AG^{1/2})\mathbf{x} = \mathbf{b}$. To see this, we note that for an underdetermined full row-rank matrix A and vector $\mathbf{b}$ we have that the minimum 2-norm solution is given by

$$\mathbf{x}^* = X^\top (XX^\top)^{-1}\mathbf{b}. \tag{5.14}$$

In this case, we set $X := AG^{1/2}$ to retrieve our result.

5.3 Convergence of Adagrad

Now that we've dealt with the simple case of fixed preconditioner gradient descent, we now move onto the main case of proving convergence of Adagrad in the underdetermined LS regime (5.3) under update rule (5.5). Unlike fixed preconditioner GD, we do not supply a result similar to theorem (5.2.1) for Adagrad.

The following two results are some preliminary facts that will be used in the lemmas and theorems that follow:

Fact 5.3.1. *Let $A \in \mathbb{R}^{n\times d}$. The eigenvalues of A^TA and AA^T are nonnegative.*

Corollary 5.3.1.1. *Let $A \in \mathbb{R}^{n\times d}$, and let $M \in \mathbb{R}^{n\times n}$ be a diagonal positive semi-definite matrix. The eigenvalues of AMA^T and A^TMA are nonnegative.*

Proof. We prove the case for AMA^T. The work for A^TMA is identical with a difference in transpose placement.

Since M is a positive semi-definite diagonal matrix (i.e. its diagonal entries are all nonnegative), we may write $M = M^{1/2}M^{1/2}$. Then,

$$\begin{aligned} AMA^T &= (AM^{1/2})(M^{1/2}A^T) \\ &= (AM^{1/2})(AM^{1/2})^T \qquad\qquad [M \text{ symmetric}] \end{aligned}$$

Letting $B = (AM^{1/2})$ we get

$$AMA^T = BB^T$$

we conclude by fact 5.3.1 □

From here, we work with a recurrence relation derived based on the update (5.4) where the diagonal entries of the diagonal matrix $(G_k^{1/2})^{-1}$ are described by (5.5). Let $\mathbf{u}_k = A\mathbf{x}_k - \mathbf{b}$. For simplicity, let's rewrite G_k as:

$$G_k := \frac{\eta}{\sqrt{\hat{G}_k}}, \tag{5.15}$$

where $\hat{G}_k$ is a diagonal matrix where the i-th diagonal entry is given by

$$\hat{G}_{k_{(i,i)}} = \sum_{l=0}^{k} (A_i^T \mathbf{u}_l)^2 + k\epsilon. \tag{5.16}$$

Here the reciprocal and square root operations are applied entry wise. Here, we combine the step-size η into G_k because our following results hold regardless of our choice of $\eta > 0$, and thus we combine it into G_k to make the formulations easier to read. We may thus rewrite (5.4) as

$$\begin{aligned} \mathbf{x}_{t+1} &= \mathbf{x}_k - G_k \nabla f(\mathbf{x}_k) \\ \implies A\mathbf{x}_{k+1} - \mathbf{b} &= A\mathbf{x}_k - \mathbf{b} - AG_k \nabla f(\mathbf{x}_k) \\ \implies A\mathbf{x}_{k+1} - \mathbf{b} &= A\mathbf{x}_k - \mathbf{b} - AG_k A^\top (A\mathbf{x}_k - \mathbf{b}) \\ \implies A\mathbf{x}_{k+1} - \mathbf{b} &= (I - AG_k A^\top)(A\mathbf{x}_k - \mathbf{b}) \\ \implies \mathbf{u}_{k+1} &= (I - AG_k A^\top)\mathbf{u}_k \end{aligned} \tag{5.17}$$

where the matrix $(I - \eta AG_k A^\top)$ enjoys the property of being symmetric. Lemma 5.3.2 will allow us to guarantee that after some finite iteration of k, the matrix $(I - AG_k A^\top)$ will have its spectral norm contained in the interval between 0 and 1.

Lemma 5.3.2. *Let $\|\cdot\|_2$ denote the spectral norm of a matrix, and, for fixed $\eta > 0$, let G_k be as in (5.15). Then, $\exists k^* \in \mathbb{N}$ such that $\forall k \geq k^*$ the symmetric matrix $(I - AG_kA^\top)$ satisfies*

$$0 < \|I - AG_kA^\top\|_2 \leq 1 - \frac{a}{\sqrt{k}} < 1$$

for some constant $a > 0$.

Proof. First, by Corollary (5.3.1.1), (AG_kA^T) is positive semi-definite. Next, by the construction of $\hat{G}_k$, it continually grows by (at least) an $\epsilon > 0$ per iteration and is therefore unbounded. Thus, $\exists k^* \in \mathbb{N}$ such that $\forall k \geq k^*$, the diagonal elements of $\hat{G}_k$ satisfy

$$|\hat{G}_{k_{(i,i)}}| \geq 4\eta \cdot \|AA^\top\|_2. \tag{5.18}$$

The choice of lower bound on $|\hat{G}_{k_{(i,i)}}|$ is chosen to ensure that $(\forall k \geq k^*)$ $AG_kA^\top \prec I$. We show this by showing that $(\forall k \geq k^*)$ $\|AG_kA^\top)\|_2 < 1$. By (5.18) we have that

$$\begin{aligned}
\left\|AG_kA^\top\right\|_2 &\leq \left\|\left(\frac{1}{4\|AA^\top\|_2^2}\right)^{1/2} AA^\top\right\|_2 && (5.19)\\
&= \left(\frac{1}{4\|AA^\top\|_2^2}\right)^{1/2} \left\|AA^\top\right\|_2 && (5.20)\\
&= \left(\frac{1}{2\|A\|_2^2}\right) \|A\|_2^2 && (5.21)\\
&= \frac{1}{2} < 1 && (5.22)
\end{aligned}$$

where the inequality in (5.19) holds since, by Lemma (5.3.1), $\left(\frac{1}{2\|A\|_2^2}\right) AA^\top$ is positive semi-definite and

$$\begin{aligned}
AG_kA^T \preceq \left(\frac{1}{2\cdot\|A\|_2^2}\right) AA^\top &\iff AG_kA^\top - \left(\frac{1}{2\cdot\|A\|_2^2}\right) AA^\top \preceq \mathbf{0} && (5.23)\\
&\iff A(G_k - \left(\frac{1}{2\cdot\|A\|_2^2}\right) I)A^\top \preceq \mathbf{0} && (5.24)\\
&\iff \forall i, \quad G_{k_{(i,i)}} - \left(\frac{1}{2\cdot\|A\|_2^2}\right) < 0 && (5.25)
\end{aligned}$$

where (5.25) is satisfied by (5.18). From here, we have that

$$
\begin{aligned}
&(\forall k \geq k^*) \ \ \|AG_kA^\top)\|_2 < 1. && (5.26)\\
\implies &(\forall k \geq k^*) \ \ \|\mathbf{u}_{k+1}\|_2 \leq \|\mathbf{u}_k\|_2 && (5.27)\\
\implies &(\forall k \geq k^*) \ \ (\exists s > 0) \ \ s.t \ \ (A_i^\top \mathbf{u}_k)^2 \leq s \ \ (\forall i) && (5.28)\\
\implies &(\forall k \geq k^*) \ \ \hat{G}_{k_{(i,i)}} \leq \hat{G}_{k^*_{(i,i)}} + (k-k^*)s + (k-k^*)\epsilon \ \ (\forall i) && (5.29)\\
\implies &(\forall k \geq k^*) \ \ (\exists s' \in \mathbb{R}) \ \ s.t \ \ \hat{G}_{k_{(i,i)}} \leq ks' \ \ (\forall i) && (5.30)
\end{aligned}
$$

Where, in line (5.29),

$$\hat{G}_{k^*_{(i,i)}} = \sum_{l=1}^{k^*} (A_i^\top \mathbf{u}_l)^2 + k^*\epsilon$$

and

$$(k-k^*)s \geq \sum_{l=(k^*+1)}^{k} (A_i^\top \mathbf{u}_l)^2.$$

Line (5.30) introduces a new constant $s' \in \mathbb{R}$ which can be chosen as follows, $\forall i$

$$
\begin{aligned}
G_{k_{(i,i)}} &\leq G_{k^*_{(i,i)}} + (k-k^*)s + (k-k^*)\epsilon \\
&= \sum_{l=1}^{k^*} (A_i^\top \mathbf{u}_l)^2 + k^*\epsilon + (k-k^*)s + (k-k^*)\epsilon \\
&= \left(\sum_{l=1}^{k^*} (A_i^\top \mathbf{u}_l)^2 - k^*s \right) + (ks - k\epsilon)
\end{aligned}
$$

letting constant $C = \max\left\{ \sum_{l=1}^{k^*} (A_i^\top \mathbf{u}_l)^2 - k^*s, \ 0 \right\}$ we continue to get

$$
\begin{aligned}
&\leq C + k(s+\epsilon) \\
&\leq kC + k(s+\epsilon) \\
&= k(C+s+\epsilon)
\end{aligned}
$$

finally, we set constant $s' = C + s + \epsilon$.

Taking the reciprocal square root of line (5.30) gives us $(\hat{G}_{k_{(i,i)}})^{-1/2} \geq (ks')^{-1/2}$. Then we have that

$$\begin{aligned}
&(\forall k \geq k^*) \;\; \mathbf{0} \prec (a \cdot k^{-1/2})I \preceq AG_kA^\top \prec I \\
\implies &(\forall k \geq k^*) \;\; \mathbf{0} \succ -(a \cdot k^{-1/2})I \succeq -AG_kA^\top \succ -I \\
\implies &(\forall k \geq k^*) \;\; I \succ I - (a \cdot k^{-1/2})I \succeq I - AG_kA^\top \succ \mathbf{0} \\
\implies &(\forall k \geq k^*) \;\; 1 > 1 - (a \cdot k^{-1/2}) \geq \|I - AG_kA^\top\|_2 > 0
\end{aligned}$$

where $a := (s')^{-1/2} \in \mathbb{R}$ a constant.

This concludes the proof. $\square$

The reason we need lemma 5.3.2 is to ensure that, in the limit, the recurrence remains a contraction at each iteration. For the iterations $k < k^*$ there is no control on the bound. With this result, we may prove the following lemma:

Lemma 5.3.3. *Given the recurrence* (5.17), *we have that*

$$\mathbf{u}_\infty := \lim_{k\to\infty} \mathbf{u}_k = \prod_{k=1}^{\infty}(I - AG_kA^\top)\mathbf{u}_0 = \mathbf{0}. \tag{5.31}$$

Proof. The left hand side of (5.31) comes directly from the derived recurrence itself

$$\begin{aligned}
\mathbf{u}_k &= (I - AG_{k-1}A^\top)\mathbf{u}_{k-1} \\
&= (I - AG_{k-1}A^\top) \cdot ... \cdot (I - AG_1A^\top)\mathbf{u}_0 \\
&= \prod_{j=1}^{k-1}(I - AG_jA^\top)\mathbf{u}_0 \\
\implies \lim_{k\to\infty} \mathbf{u}_k &= \prod_{j=1}^{\infty}(I - AG_jA^\top)\mathbf{u}_0.
\end{aligned}$$

To show the equality of (5.31), it suffices to show that

$$\prod_{k=1}^{\infty} \|I - AG_kA^\top\|_2 = 0 \tag{5.32}$$

since we have that

$$\|\mathbf{u}_\infty\|_2 \leq \prod_{k=1}^{\infty} \|I - AG_kA^\top\|_2 \cdot \|\mathbf{u}_0\|_2.$$

We show (5.32) by applying the infinite product theorem and Lemma (5.3.2). The infinite product theorem states that if $\{a_n\}_{n=0}^{\infty}$ is a sequence such that $(\forall n)\ 0 < a_n < 1$ then

$$\prod_{n=1}^{\infty}(1-a_n) \to 0 \iff \sum_{n=1}^{\infty} a_n = \infty.$$

In our case, Lemma (5.3.2) tells us that $(\exists k^*)$ s.t $(\forall k \geq k^*)$ $(\exists a > 0)$ constant such that

$$0 < \|I - AG_kA^\top\|_2 \leq (1 - a/\sqrt{k}) < 1. \tag{5.33}$$

Using the infinite product theorem while setting $a_n = a/\sqrt{n}$ $(\in (0,1)\ (\forall n \geq k^*))$, we have that

$$\sum_{n=k^*}^{\infty} \frac{a}{\sqrt{n}} = \infty \tag{5.34}$$

which is true due to the integral test for infinite sums

$$\begin{aligned} &\int_{k^*}^{\infty} \frac{a}{\sqrt{x}} dx = \left[2a\sqrt{x}\right]_{k^*}^{\infty} = \infty \\ \implies &\sum_{n=k^*}^{\infty} \frac{a}{\sqrt{n}} = \infty. \end{aligned}$$

So we may conclude that

$$\begin{aligned} &\prod_{n=k^*}^{\infty} \|I - AG_kA^\top)\|_2 \leq \prod_{n=k^*}^{\infty} (1 - a/\sqrt{n}) = 0 \\ \implies &\prod_{k=1}^{\infty} \|I - AG_kA^\top)\|_2 = 0 \end{aligned}$$

This concludes the proof. □

Lemma 5.3.3 tells us that under the equivalent recurrence (5.17) to the Adagrad update (5.4), the limiting value of $\|\mathbf{u}_k\|$ converges to 0 which implies that $\|A\mathbf{x}_k - \mathbf{b}\|$ converges to 0 in the limit. This, however, is not enough to conclude convergence of Adagrad. We now need to show the existence of some $\mathbf{x}^*$ such that $\mathbf{x}_k \to \mathbf{x}^*$ as $k \to \infty$. From this, we can conclude by lemma 5.3.3 that Adagrad reaches an $\mathbf{x}^*$ such that $A\mathbf{x}^* = \mathbf{b}$. This is bundled into theorem 5.3.6. Before we get to that theorem, however, we require the result presented in Lemma 5.3.5. To prove this lemma requires the following result:

Lemma 5.3.4. *For all $k, p \in \mathbb{N}$ satisfying $k \geq 2$ and $p > 1$, the following inequality holds:*

$$(k-1)^p \geq k^p - pk^{p-1}. \tag{5.35}$$

Proof. Dividing both sides of (5.35) by k, and letting $x := 1/k$ we have that

$$\begin{aligned}
& x\left(\frac{1}{x}-1\right)^p \geq \left(\frac{1}{x}\right)^{p-1} - p\left(\frac{1}{x}\right)^{p-2} \\
\iff & x^p\left(\frac{1}{x}-1\right)^p \geq x^{p-1}\left(\frac{1}{x}\right)^{p-1} - px^{p-1}\left(\frac{1}{x}\right)^{p-2} \\
\iff & (1-x)^p \geq 1 - px \\
\iff & (1-x)^p - (1-px) \geq 0. && (\dagger)
\end{aligned}$$

To show that † holds, we observe that the function value at $x = 0$ is 0, and consider the first and second derivatives:

$$\frac{d}{dx}\dagger(x) = p(1-x)^{p-1} - p \tag{5.36}$$

$$\frac{d^2}{dx^2}\dagger(x) = p(p-1)(1-x)^{p-2}. \tag{5.37}$$

When $x = 0$, the first derivative also evaluates to 0 implying a critical point. For $p > 1$ and $|x| < 1$ we have that the second derivative is positive. All together, we have that for $p > 1$ and $|x| < 1$, † holds. Substituting k back, we conclude that (5.35) holds for $k \geq 2$ and $p > 1$.

This concludes the proof. □

We now state and prove Lemma 5.3.5:

Lemma 5.3.5. *Let $a \in \mathbb{R}$ and $\hat{k}_* \in \mathbb{N}$ be the constant and iteration (respectively) in which the results from lemma 5.3.2 hold. Let $k_* \in \mathbb{N}$ be such that $k_* \geq \hat{k}_*$ and $a\sqrt{k_*} \geq 3$. Choose $b \in \mathbb{R}$ satisfying*

$$b \geq k_*^p \cdot \left(\prod_{n=1}^{k_*}\left(1 - \frac{a}{\sqrt{n}}\right)\right) \tag{5.38}$$

for fixed $p \in \mathbb{R}$ satisfying $1 < p < a\sqrt{k_*}$. Then, $\forall k \geq k_*$, the following bound holds:

$$\prod_{n=1}^{k}\left(1 - \frac{a}{\sqrt{n}}\right) \leq \frac{b}{k^p}. \tag{5.39}$$

Proof. We show this holds by induction. We begin with the base case: $k = k_*$. Clearly, by our choice of b, we must have that

$$\prod_{n=1}^{k_*} \left(1 - \frac{a}{\sqrt{n}}\right) \leq \frac{b}{k_*^p}. \ \checkmark \tag{5.40}$$

Now, assume the hypothesis holds for all integers from k_* up to and including $(k-1)$ where $(k-1) \geq k_*$. We show that the hypothesis holds for $k \geq k^*$:

$$\begin{aligned}
\prod_{n=1}^{k} \left(1 - \frac{a}{\sqrt{n}}\right) &= \prod_{n=1}^{k-1} \left(1 - \frac{a}{\sqrt{n}}\right) \cdot \left(1 - \frac{a}{\sqrt{k}}\right) \\
&\leq \frac{b}{(k-1)^p} \cdot \left(1 - \frac{a}{\sqrt{k}}\right) && \text{(induction hypothesis)} \\
&\leq \frac{b}{k^p - pk^{p-1}} \cdot \left(1 - \frac{a}{\sqrt{k}}\right) && \text{(by lemma 5.3.4)} \\
&= \frac{b}{k^p - pk^{p-1}} - \frac{ab}{\sqrt{k}\,(k^p - pk^{p-1})} \\
&= \frac{\sqrt{k}b - ab}{\sqrt{k}\,(k^p - pk^{p-1})} \\
&= \frac{\sqrt{k}\left(1 - \frac{a}{\sqrt{k}}\right) b}{\sqrt{k}\,(k^p - pk^{p-1})} \\
&= \frac{\left(1 - \frac{a}{\sqrt{k}}\right) b}{\left(1 - \frac{p}{k}\right) k^p}
\end{aligned}$$

$$\begin{aligned}
&= \left(\frac{1 - \frac{a}{\sqrt{k}}}{1 - \frac{p}{k}}\right) \frac{b}{k^p} \\
&\leq \frac{b}{k^p}. && \text{(by choice of } p\text{)}
\end{aligned}$$

By induction, we conclude the proof. □

Lemma 5.3.5 tells us that the sequence $\prod_{n=1}^{k} \left(1 - \frac{a}{\sqrt{n}}\right)$, with respect to k, converges to zero as fast as (or faster than) the sequence $\frac{b}{k^p}$. We now present the convergence theorem for Adagrad in the underdetermined linear regression regime:

Theorem 5.3.6. *Let $f(\mathbf{x})$ be as in (5.2) where $A \in \mathbb{R}^{n \times d}$ $(n < d)$, fix $\eta > 0$, $\mathbf{x} \in \mathbb{R}^d$, and $\mathbf{b} \in \mathbb{R}^n$. Let $G_k \in \mathbb{R}^{d \times d}$ be the Adagrad preconditioner matrix (5.15). Consider the Adagrad update*

$$\mathbf{x}_{k+1} = \mathbf{x}_k - G_k \nabla f(\mathbf{x}_k), \tag{5.41}$$

with inital value $\mathbf{x}_0 = \mathbf{0}$. Then there exists an $\mathbf{x}^* \in \arg\min_{\mathbf{x} \in \mathbb{R}^d} \frac{1}{2}\|A\mathbf{x} - \mathbf{b}\|_2^2$ such that $\|\mathbf{x}_k - \mathbf{x}^*\| \to 0$ as $k \to \infty$.

Proof. In order to prove this, we need to show two things. First we need to show existence of some point $\mathbf{x}^* \in \mathbb{R}^d$ such that $\|\mathbf{x}_k - \mathbf{x}^*\| \to 0$ as $k \to \infty$, then we need to show that the converged point $\mathbf{x}^*$ lies in the set of solutions to (5.3). From Lemmas (5.3.2) and (5.3.3) we have that the Adagrad update ensures that $\|A\mathbf{x}_k - \mathbf{b}\| \to 0$ as $k \to \infty$. Therefore, if we can show that there exists some $\mathbf{x}^*$ such that $\|\mathbf{x}_k - \mathbf{x}^*\| \to 0$ as $k \to \infty$ we are done.

Let $\mathbf{u}_k = A\mathbf{x}_k - \mathbf{b}$. From the Adagrad update (5.41), we can unravel the recurrence to get the following representation for $\mathbf{x}_k$:

$$\begin{aligned} \mathbf{x}_k &= \mathbf{x}_{k-1} - G_{k-1}A^\top \mathbf{u}_{k-1} \\ &= \mathbf{x}_0 - \sum_{i=0}^{k-1} G_i A^\top \mathbf{u}_i \\ &= -\sum_{i=0}^{k-1} G_i A^\top \mathbf{u}_i. \end{aligned} \tag{5.42}$$

To show that this sum converges when taking the limit as $k \to \infty$, it is enough to show that

$$\|G_k A^\top \mathbf{u}_k\| \to 0 \text{ as k} \to \infty \tag{5.43}$$

faster than $\frac{b}{k^p}$ for some $b \in \mathbb{R}$ and $p > 1$. Here we denote $\|\cdot\|$ as the 2-norm. To show this, we simply note that

$$\|G_k A^\top \mathbf{u}_k\| \leq \|G_k\| \cdot \|A\| \cdot \|\mathbf{u}_k\| \leq \|G_0\| \cdot \|A\| \cdot \|\mathbf{u}_k\|. \tag{5.44}$$

Let $\hat{k}^* \in \mathbb{N}$ and $a \in \mathbb{R}$ be the iterate and constant, respectively, that satisfy Lemma 5.3.2, and let $k^* \in \mathbb{N}$ such that $k^* \geq \hat{k}^*$ and $a\sqrt{k^*} \geq 3$. Let $b' \in \mathbb{R}$ and $p \in \mathbb{R}$ be chosen as in Lemma 5.3.5, and let

$$b := \max\left\{b', (\|G_0\| \cdot \|A\| \cdot \|\mathbf{u}_0\|) \cdot b'\right\}. \tag{5.45}$$

Then $\forall k \geq k^*$, we have that

$$\begin{aligned}
\|G_0\| \cdot \|A\| \cdot \|\mathbf{u}_k\| &\leq (\|G_0\| \cdot \|A\| \cdot \|\mathbf{u}_0\|) \prod_{n=1}^{k} \left(1 - \frac{a}{\sqrt{n}}\right) && \text{(by Lemma 5.3.2)} \\
&\leq \|G_0\| \cdot \|A\| \cdot \|\mathbf{u}_0\| \cdot \frac{b'}{k^p} && \text{(by Lemma 5.3.5)} \\
&\leq \frac{b}{k^p}.
\end{aligned}$$

Due to such an upper bound, we have shown that,

$$\lim_{k \to \infty} \frac{\|G_k A^T u_k\|}{\left(\frac{b}{k^p}\right)} \leq 1 \tag{5.46}$$

and therefore, $\|G_k A^T u_k\|$ converges to 0 faster than b/k^p. Therefore, there exists an $\mathbf{x}^*$ such that

$$\lim_{k \to \infty} \mathbf{x}_k = \mathbf{x}^*$$

.
This concludes the proof. □

We conclude this section by noting that, unlike the fixed preconditioner GD scenario, we do not have an explicit representation of $\mathbf{x}^*$. In the next section, however, we look into a property that $\mathbf{x}^*$ has when attained via Adagrad, and in Section 5.6 we present a solution for Adagrad in the over-parameterized binary least-squares classification regime from the work by Wilson, et al. [57].

As was mentioned earlier in Chapter 2, there exists recent work on the convergence of both Adagrad and Adam in a more general regime [20]. We note that the discovery of this work came only after our own analysis above, however, we still summarise the authors' results briefly here.

In their paper, the authors provide a proof of convergence for both Adam and Adagrad in the regime of smooth (possibly non-convex) objective functions with bounded gradient. Proof of convergence is also shown for heavy-ball momentum variants of both Adagrad and Adam. Let $N \in \mathbb{N}^*$ be a certain number of iterations and denote τ_N to be a random index with value in $\{0, \ldots, N-1\}$, so that

$$\forall j \in \mathbb{N},\ j < N.\ \mathbb{P}[\tau = j] \propto 1 - \beta_1^{N-j} \tag{5.47}$$

where $0 < \beta_1 < 1$ is the same β_1 parameter used in Adam seen in Section 4.4. In order to cover their results, we must also consider three assumptions made by the authors: First, they assume that the objective function F is bounded below by F_*, i.e., $\forall \mathbf{x} \in \mathbb{R}^d$, $F(\mathbf{x}) \geq F_*$; Second, they assume that the ℓ_∞-norm of the stochastic gradients is uniformly almost surely bounded, that is, there is an $R > \sqrt{\epsilon}$ so that

$$\forall \mathbf{x} \in \mathbb{R}^d, \quad \|\nabla F(\mathbf{x})\|_\infty \leq R - \sqrt{\epsilon} \tag{5.48}$$

almost surely; Finally, they assume smoothness of the objective function, i.e., its gradient is L-Lipschitz continuous with respect to the ℓ_2-norm.

With the notation defined for τ and the assumptions just posed, the authors' propose the following convergence result which pertains to the convergence of Adagrad in a non-momentum setting:

Theorem 5.3.7 (Adagrad). *Given the three assumptions above, τ defined as in (5.47), and the iterates $\mathbf{x}_n \in \mathbb{R}^d$ generated by the Adagrad update with constant step-size $\alpha_n \equiv \alpha > 0$, we have for any $N \in \mathbb{N}^*$,*

$$\mathbb{E}\left[\|\nabla F(\mathbf{x}_\tau)\|^2\right] \leq 2R\frac{F(\mathbf{x}_0) - F_*}{\alpha\sqrt{N}} + \frac{1}{\sqrt{N}}(4dR^2 + \alpha dRL)\ln\left(1 + \frac{NR^2}{\epsilon}\right). \tag{5.49}$$

Proof. See [20]. □

While this result provides a convergence result for Adagrad in a more general regime of study, we contrast the results present in [20] from our own by noting that the results pertain to the convergence of the function values to the optimal. In our proof of the over-parameterized least-squares regime, we show that the iterates of Adagrad themselves converge to an optimal solution $\mathbf{x}^*$.

5.4 Adagrad Promotes Diffuse Solutions

So far we have shown the asymptotic convergence of Adagrad as well as results related to a simple case of fixed preconditioner GD, all in the underdetermined linear regression regime. Unlike the simple case, we've yet to showcase any algorithmic behaviours of Adagrad in the underdetermined LR regime. In this section we flesh out an experimental behaviour we encountered pertaining to Adagrad in this regime. While the discussion and the experimental results are left to the following chapter, we introduce our finding here and present an analytical result which gives us some theoretical guarantee of the behaviour under certain assumptions.

In short, what we encountered experimentally was that under problem (5.3), with a step size, $\eta > 0$ small enough, we notice that, compared to standard GD and SGD, the solution to (5.3) produced by Adagrad, when $\mathbf{x}_0 = \mathbf{0}$, is noticeably more equally balanced. That is, if we think of the coordinates of the Adagrad solution $\mathbf{x}^*$ to be the weights assigned to each feature to determine an output, then the weights tend to be closer together in absolute value than GD and SGD. Before we jump into the details, we begin with some motivating results which guided us to running such an experiment in the first place.

It is common knowledge that when running (stochastic) gradient descent in the underdetermined LR regime with an initial value of $x_0 = \mathbf{0}$, that the algorithms converge to the minimum two-norm (or least two-norm) solution. That is, out of the infinite number of solutions, both algorithms bias towards the solution $\mathbf{x}^*$ satisfying:

$$\begin{aligned} &\min_{\mathbf{x}\in\mathbb{R}^d} \frac{1}{2}\|\mathbf{x}\|_2^2 \\ &s.t \;\; A\mathbf{x} = \mathbf{b}. \end{aligned} \tag{5.50}$$

where $A \in \mathbb{R}^{n\times d}$ is underdetermined (i.e. $n < d$), and $\mathbf{b} \in \mathbb{R}^n$. To see this, consider the following special case of Theorem 5.2.1:

Lemma 5.4.1. *Let $A \in \mathbb{R}^{n\times d}$, $n < d$ and full row rank, $\mathbf{x} \in \mathbb{R}^d$, and $\mathbf{b} \in \mathbb{R}^n$. Consider the minimum 2-norm optimization problem (5.50); then a solution to such a problem is*

$$\mathbf{x}^* = A^\top (AA^\top)^{-1}\mathbf{b}. \tag{5.51}$$

To see why Lemma 5.4.1 is a special case of Theorem 5.2.1, set the preconditioner in 5.2.1 to $G = I$, the identity matrix, then we recover (5.51). This confirms that GD converges to the minimum two-norm solution when the initial point $\mathbf{x}_0 = \mathbf{0}$.

The follow up to such a result would be to understand what sort of solution Adagrad converges to when $\mathbf{x}_0 = \mathbf{0}$. Does its solution have any relation to the minimum norm solution as did GD? After running experiments, it turns out that, in general, it does not converge to the minimum two-norm solution. Instead what we noticed while running our experiments was that if we chose the step size η to be small ($\eta \leq 0.01$), then when compared with the weights assigned to each feature generated by the minimum two-norm solution, the weights assigned to each feature generated by the Adagrad solution were more diffuse.

To concretely describe this behaviour, we used the following metric as a way to compare solutions of algorithms when running on the same problem.

Definition 5.4.1 (The Local-Sparsity Metric (LSM))**.** Given a vector $\mathbf{x} \in \mathbb{R}^d \setminus \{\mathbf{0}\}$, we measure its local sparsity, i.e. how similar its coordinates are to one another, based on the following metric $LSM : \mathbb{R}^d \setminus \{\mathbf{0}\} \to \mathbb{R}$ where

$$LSM(\mathbf{x}) := \frac{\|\mathbf{x}\|_2}{\|\mathbf{x}\|_1}. \tag{5.52}$$

This metric and the theory on the relationship between the 2-norm and 1-norm arises in the theory of compressive sensing covered by a Russian paper written by Garnaev and Gluskin [22], and quoted in English by Zhang in his tech report [65]. Compressive sensing deals with the problem of finding the sparsest solution to an underdetermined system of linear equations. Our observations show that Adagrad promotes solutions opposite to what is trying to be achieved in compressive sensing. LSM captures approximate sparsity, or, in other words, localization of a vector's coordinates.

To better grasp what is approximate sparsity/locality, we consider the following example: Two algorithms, algo. 1 and algo. 2, are run on the same loss function of a regression task in order to generate an optimal set of weights, which we'll denote $\mathbf{w}_1$ and $\mathbf{w}_2$ respectively. The weights designate the importance of each feature when generating a result. The following optimal weights are produced:

$$\mathbf{w}_1 = (0.25, 0.25, 0.15, 0.35)^\top$$
$$\mathbf{w}_2 = (0.90, 0.02, 0.02, 0.06)^\top.$$

If we look at the weights of $\mathbf{w}_1$, we see that the total weight is diffuse pretty evenly among each of the coordinates. That is, the solution is not approximately sparse; each feature has similar power in determining the output. When we compare this to $\mathbf{w}_2$, we see that a large portion of the weight is given to the first feature compared to the other features. That is, the coordinates are approximately sparse with respect to the first coordinate, or, in other words, $\mathbf{w}_2$ has strong locality at the first coordinate, and is not diffuse like $\mathbf{w}_1$. Using the LSM, we should have that $LSM(\mathbf{w}_2) > LSM(\mathbf{w}_1)$, i.e. $\mathbf{w}_2$ is more approximately sparse than $\mathbf{w}_1$. Indeed we have

$$LSM(\mathbf{w}_2) \approx 0.90244 > 0.51962 \approx LSM(\mathbf{w}_1).$$

Lemma 5.4.2. *For any vector $\mathbf{x} \in \mathbb{R}^d \setminus \{\mathbf{0}\}$, its LSM is bounded as follows:*

$$\frac{1}{\sqrt{d}} \leq LSM(\mathbf{x}) \leq 1. \tag{5.53}$$

Proof. We note that by the sign invariance under the norms, it suffices to prove this for $\mathbf{x} \geq \mathbf{0}$. So, let $\mathbf{x} = (x_1, \ldots, x_d)^\top > \mathbf{0}$. To show the lower bound, we have by the Cauchy-Schwartz inequality that

$$\begin{aligned} \|\mathbf{x}\|_1 = \sum_{k=1}^{d} |x_k| \leq \left(\sum_{k=1}^{d} |x_k|^2\right)^{1/2} \cdot \left(\sum_{k=1}^{d} 1^2\right)^{1/2} = \sqrt{d}\|\mathbf{x}\|_2 \\ \implies \frac{1}{\sqrt{d}} \leq \frac{\|\mathbf{x}\|_2}{\|\mathbf{x}\|_1}. \end{aligned} \tag{5.54}$$

For the upper bound, squaring both sides we get that

$$\|\mathbf{x}\|_1^2 = (x_1 + \cdots + x_d)^2 \geq x_1^2 + \cdots + x_d^2 = \|\mathbf{x}\|_2^2 \tag{5.55}$$

which holds due to our assumption that $\mathbf{x} \geq \mathbf{0}$. Rearranging yields our upper bound:

$$\|\mathbf{x}\|_2^2 \leq \|\mathbf{x}\|_1^2 \implies \frac{\|\mathbf{x}\|_2}{\|\mathbf{x}\|_1} \leq 1. \tag{5.56}$$

This concludes the proof. $\square$

We can think of a vector attaining the lower bound as one which diffuses its total weight equally in absolute value to each coordinate such as $\mathbf{x} = (1, 1, -1, 1)^\top$. We can think of a vector attaining the upper bound as one in which approximate sparsity becomes true sparsity around a single coordinate such as $\mathbf{x} = (0, 0, 1, 0)^\top$. This can be seen mathematically as follows: for the lower bound, if we are given a vector $\mathbf{x} = (a, \ldots, a)^\top \in \mathbb{R}^d$, $a \neq 0$, of equal (absolute) value across each coordinate, then we get

$$LSM(\mathbf{x}) = \frac{\|\mathbf{x}\|_2}{\|\mathbf{x}\|_1} = \frac{\sqrt{a^2 + \cdots + a^2}}{|a| + \cdots + |a|} = \frac{\sqrt{d}|a|}{d|a|} = \frac{\sqrt{d}}{d} = \frac{1}{\sqrt{d}}. \tag{5.57}$$

For the upper bound, if given a vector $\mathbf{x} = (a, 0, \dots, 0)^\top \in \mathbb{R}^d$, $a \neq 0$, then we get

$$LSM(\mathbf{x}) = \frac{\|\mathbf{x}\|_2}{\|\mathbf{x}\|_1} = \frac{\sqrt{a^2}}{|a|} = 1 \tag{5.58}$$

where the coordinate in which we set a is arbitrary.

Using this metric, our experiments show that the Adagrad solution $\mathbf{x}_{ada}$ generated using a small η, yields an LSM that is closer to the lower bound than the upper bound in (5.53). Furthermore, this LSM is lower when compared against the LSM calculated using the GD solution $\mathbf{x}_{GD}$ (or, in other words, the least norm solution), that is, $LSM(\mathbf{x}_{ada}) < LSM(\mathbf{x}_{GD})$. Intuitively, this can be thought of as Adagrad promoting the coordinates of sparsely present features while suppressing the formation of locality of coordinates of non-sparse features (including when compared to GD). To further explore this in a non-rigorous manner (for now), let's look at what happens to the preconditioner of Adagrad in the face of sparse features versus non-sparse features.

Recall the Adagrad preconditioner (5.5) for each coordinate x_i of $\mathbf{x}$, $i = 1, \dots, d$. If we ignore the ϵ and focus on the main portion of the preconditioner, namely, $\sqrt{\sum_{\tau=1}^{k} (\mathbf{g}_\tau)_i^2}$, we see that if our past i-th partial derivatives are large, then the preconditioner value for that coordinate will be small since we take the reciprocal. Likewise, if our past i-th partial derivatives are small, then the preconditioner value for that coordinate will be large. If we are dealing with a sparse feature, then its gradient values are small since

$$(\mathbf{g}_k)_i = (A^\top(A\mathbf{x}_k - \mathbf{b}))_i = A(:, i)^\top(A\mathbf{x}_k - \mathbf{b})$$

where $A(:, i)$ is the sparse i-th feature column of the data matrix A. So under a sparse feature, $\|(\mathbf{g}_k)_i\|$ is small. On the contrary, if we are dealing with non-sparse features, there is a higher chance that the partial derivatives pertaining to those features are larger in value (compared to the sparse features) as there is less multiplication by 0. So in the face of sparse features, Adagrad will take larger steps in those dimensions, while for non-sparse features, Adagrad will not step as fast. In comparison, GD provides equal promotion of features since we multiply every coordinate of the gradient by the same step size parameter η. In other words, one should expect that on a given step, k, the size of the step taken in the direction of a non-sparse feature coordinate should be larger for GD when compared to Adagrad, and larger for Adagrad when compared to GD when stepping in the direction of a sparse feature coordinate.

This intuition is made concrete with the following theorem:

Theorem 5.4.3. *Let $A \in \mathbb{R}^{n \times d}$ such that $n < d$ and let $A(:,i) \in \mathbb{R}^n$ denote the i-th column of A for $i = 1, \ldots, d$. Let $\mathbf{x}_k^{ada}, \mathbf{x}_k^{gd} \in \mathbb{R}^d$ denote the k-th iterate generated by Adagrad and gradient descent, respectively, and $\mathbf{b} \in \mathbb{R}^n$. Let $f : \mathbb{R}^d \to \mathbb{R}$ be the least squares function*

$$f(\mathbf{x}) = \frac{1}{2}\|A\mathbf{x} - \mathbf{b}\|_2^2. \tag{5.59}$$

Finally, consider the magnitudes of both the Adagrad and gradient descent updates at the i-th coordinate:

$$|\Delta \mathbf{x}_k^{ada}(i)| = \eta \cdot \left| \frac{1}{\sqrt{\sum_{\tau=0}^{k}(\mathbf{g}_\tau^{ada}(i))^2 + k\epsilon}} \cdot \mathbf{g}_k^{ada}(i) \right| \tag{5.60}$$

and

$$|\Delta \mathbf{x}_k^{gd}(i)| = \eta \cdot \left| \mathbf{g}_k^{gd}(i) \right| \tag{5.61}$$

respectively, where both algorithms start at $\mathbf{x}_0 = \mathbf{0}$. Assume that there exist coordinates, i, such that

$$|A(:,i)^\top \mathbf{b}| \geq \delta \|A(:,i)\| \tag{5.62}$$

where $\delta > 0$. That is, we consider the coordinates in which $|A(:,i)^\top \mathbf{b}|$ are at least a constant factor greater than the norm of the column $\|A(:,i)\|$. Then, for $\eta > 0$ chosen small enough such that, if η_1 is such that $\|I - \eta_1 A^T A\|_2 \leq 1$, and η_2 is such that $\|I - \eta_2 G_0 A^T A\|_2 \leq 1$, we have that $\eta \leq \min\{\eta_1, \eta_2\}$, where G_0 is the Adagrad preconditioner matrix at iteration $k = 0$ where each diagonal entry is of the form (5.5) for $k = 0$, and for iterations $k \leq \min\left\{\left(\frac{\delta}{2\eta\|A\|\cdot\|A^T b\|}\right), \left(\frac{\delta}{2\eta\|A\|\cdot\|G_0\|\cdot\|A^T b\|}\right)\right\}$ the following bounds hold

$$\left| \frac{3}{\sqrt{k} \cdot \sqrt{(\frac{3}{2}(A^\top \mathbf{b})_i)^2 + \epsilon}} \right| \leq \left| \frac{\Delta \mathbf{x}_k^{ada}(i)}{\Delta \mathbf{x}_k^{gd}(i)} \right| \leq \left| \frac{3}{\sqrt{k} \cdot \sqrt{(\frac{1}{2}(A^\top \mathbf{b})_i)^2 + \epsilon}} \right| \tag{5.63}$$

for i satisfying our assumption, where $(A^\top \mathbf{b})_i$ is the i-th coordinate of the vector $A^\top \mathbf{b}$.

Proof. Under our hypothesis, there exists coordinates, i, such that we satisfy (5.62). Under such i, we note that the i-th coordinate of $\nabla f(\mathbf{x})$ is

$$\mathbf{g}(i) = (A^\top A\mathbf{x} - A^\top \mathbf{b})_i = (A^\top A\mathbf{x})_i - (A^\top \mathbf{b})_i$$

Now, for such i respecting (5.62), we must have for small enough iterations k that the following inequality holds:

$$\left|(A^\top A\mathbf{x}_k)_i\right| \leq \frac{1}{2}\left|(A^\top \mathbf{b})_i\right| \tag{5.64}$$

for both $\mathbf{x}_k^{ada}$ and $\mathbf{x}_k^{gd}$. We show this holds for both gradient descent and Adagrad under our hypothesis and assumptions. First we show this holds for gradient descent which has the following update:

$$\mathbf{x}_k = \mathbf{x}_{k-1} - \eta A^\top (A\mathbf{x}_{k-1} - \mathbf{b}).$$

This can be rewritten into the following recursive form:

$$\mathbf{x}_k = (I - \eta A^\top A)\mathbf{x}_{k-1} + \eta A^\top \mathbf{b} \tag{5.65}$$

which can be expanded and written in terms of $\mathbf{x}_0$:

$$\begin{aligned}
\mathbf{x}_k &= (I - \eta A^\top A)\mathbf{x}_0 + A^T\mathbf{b} \cdot \sum_{i=1}^{k-1}(I - \eta A^\top A)^i && (5.66)\\
&= A^\top \mathbf{b} \cdot \sum_{i=0}^{k-1}(I - \eta A^\top A)^i && (5.67)
\end{aligned}$$

since $\mathbf{x}_0 = 0$. Applying the 2-norm on both sides, we have

$$\begin{aligned}
\|\mathbf{x}_k\| &= \eta\|A^\top \mathbf{b} \cdot \sum_{i=0}^{k-1}(I - \eta A^\top A)^i\| && (5.68)\\
&\leq \eta\|A^\top \mathbf{b}\| \cdot \sum_{i=0}^{k-1}\|I - \eta A^\top A\|^i && (5.69)\\
&\leq \eta\|A^\top \mathbf{b}\| \cdot \sum_{i=0}^{k-1} 1 && (5.70)\\
&= \eta k\|A^\top \mathbf{b}\|. && (5.71)
\end{aligned}$$

where the inequality from (5.69) to (5.70) is due to our choice of η. Now, for i satisfying our assumption, we have

$$\begin{aligned} |(A^\top A\mathbf{x}_k)_i| &= |A(:,i)^\top(A\mathbf{x}_k)| && (5.72)\\ &\le \|A(:,i)\|\cdot\|A\|\cdot\|\mathbf{x}_k\| && (5.73)\\ &\le \frac{1}{\delta}|A(:,i)^\top\mathbf{b}|\cdot\|A\|\cdot k\eta\|A^\top\mathbf{b}\| && (5.74)\\ &= k\cdot\left(\frac{\eta\|A\|\cdot\|A^\top\mathbf{b}\|}{\delta}\right)|A(:,i)^\top\mathbf{b}|. && (5.75)\end{aligned}$$

By our choice of k in the hypothesis, we must have that

$$|(A^\top A\mathbf{x}_k)_i| \le k\cdot\left(\frac{\eta\|A\|\cdot\|A^\top\mathbf{b}\|}{\delta}\right)|A(:,i)^\top\mathbf{b}| \le \frac{1}{2}|A(:,i)^\top\mathbf{b}|. \tag{5.76}$$

Now we show the case for Adagrad. Recall the Adagrad update to be:

$$\mathbf{x}_k = \mathbf{x}_{k-1} - \eta G_{k-1}A^\top(A\mathbf{x}_{k-1}-\mathbf{b}) \tag{5.77}$$

where G_k is the diagonal matrix with diagonal entires (5.5). Then similar to gradient descent we can express the update as a recursion:

$$\mathbf{x}_k = (I-\eta G_k A^\top A)\mathbf{x}_{k-1} + \eta G_k A^\top\mathbf{b} \tag{5.78}$$

and can be expanded to be written in terms of $\mathbf{x}_0$

$$\begin{aligned} \mathbf{x}_k &= (I-\eta G_k A^\top A)\mathbf{x}_0 + \eta G_k A^\top\mathbf{b}\cdot\sum_{i=0}^{k-1}(I-\eta G_k A^\top A)^i && (5.79)\\ &= \eta G_k A^\top\mathbf{b}\cdot\sum_{i=0}^{k-1}(I-\eta G_k A^\top A)^i && (5.80)\end{aligned}$$

where the last equality comes from the fact that $\mathbf{x}_0 = \mathbf{0}$. From here, we apply the 2-norm to get the following bound on $\|\mathbf{x}_k\|$:

$$\|\mathbf{x}_k\| \le \eta\|G_k A^\top\mathbf{b}\|\cdot\sum_{i=0}^{k-1}\|I-\eta G_k A^\top A\|^i. \tag{5.81}$$

By definition of G_k, we must have $\|G_0\| \geq \|G_k\|$ for all $k > 0$ since the denominator of any diagonal entry of G_k grows by at least $k\epsilon$. Therefore, by our choice of η we continue to have

$$\leq \eta \|G_k A^\top \mathbf{b}\| \cdot \sum_{i=0}^{k-1} \|I - \eta G_0 A^\top A\|^i \tag{5.82}$$

$$\leq \eta \|G_k A^\top \mathbf{b}\| \cdot \sum_{i=0}^{k-1} 1 \tag{5.83}$$

$$= \eta(k-1)\|G_k A^\top \mathbf{b}\| \tag{5.84}$$

$$\leq \eta k \|G_k\| \cdot \|A^\top \mathbf{b}\| \tag{5.85}$$

$$\leq \eta k \|G_0\| \cdot \|A^\top \mathbf{b}\|. \tag{5.86}$$

So by similar manipulation as in the gradient descent case, we have that

$$|(A^\top A\mathbf{x}_k)_i| \leq \|A(:,i)\| \cdot \|A\| \cdot \|x_k\| \tag{5.87}$$

$$\leq \frac{1}{\delta}|A(:,i)^\top \mathbf{b}| \cdot \|A\| \cdot k\eta\|G_0\| \cdot \|A^\top \mathbf{b}\| \tag{5.88}$$

$$= k \cdot \left(\frac{\eta\|A\| \cdot \|G_0\| \cdot \|A^\top \mathbf{b}\|}{\delta}\right) |A(:,i)^\top \mathbf{b}|. \tag{5.89}$$

Again, by our choice of k in the hypothesis, we must have that

$$|(A^\top A\mathbf{x}_k)_i| \leq \frac{1}{2}|A(:,i)^\top \mathbf{b}|. \tag{5.90}$$

This concludes the work and we continue with the main proof: Using (5.64), we derive the following upper and lower bounds for both $|\mathbf{g}_k^{ada}(i)|$ and $|\mathbf{g}_k^{gd}(i)|$:

$$\begin{aligned} |\mathbf{g}_k(i)| &= |(A^T A\mathbf{x}_k)_i - (A^T\mathbf{b})_i| \\ &\leq |(A^T A\mathbf{x}_k)_i| + |(A^T\mathbf{b})_i| \\ &\leq \frac{1}{2}|(A^T\mathbf{b})_i| + |(A^T\mathbf{b})_i| \\ &= \frac{3}{2}|(A^T\mathbf{b})_i| \end{aligned} \tag{5.91}$$

and

$$\begin{aligned}
|\mathbf{g}_k(i)| &= |(A^T A\mathbf{x}_k)_i - (A^T\mathbf{b})_i| \\
&\geq \big||(A^T\mathbf{b})_i| - |(A^T A\mathbf{x}_k)_i|\big| \\
&\geq \left||(A^T\mathbf{b})_i| - \frac{1}{2}|(A^T\mathbf{b})_i|\right| \\
&= \frac{1}{2}|(A^T\mathbf{b})_i|
\end{aligned} \tag{5.92}$$

where $\mathbf{x}_k$ depends on whether we set $\mathbf{g}_k(i)$ to be with respect to Adagrad or GD. Combining (5.92) and (5.91) we get:

$$\frac{1}{2}|(A^T\mathbf{b})_i| \leq |\mathbf{g}_k(i)| \leq \frac{3}{2}|(A^T\mathbf{b})_i|. \tag{5.93}$$

We therefore have an upper and lower bound for $\Delta\mathbf{x}_k^{gd}(i)$ with respect to $|(A^T\mathbf{b})_i|$:

$$\frac{\eta}{2}|(A^T\mathbf{b})_i| \leq \left|\Delta\mathbf{x}_k^{gd}(i)\right| \leq \frac{3\eta}{2}|(A^T\mathbf{b})_i|. \tag{5.94}$$

Now consider the denominator of (5.5) for the diagonal entries i satisfying our assumption, and denote it as $(G_k^{1/2})_i$:

$$(G_k^{1/2})_i := \sqrt{\sum_{\tau=0}^{k} \mathbf{g}_{\tau,i}^2 + k\epsilon}. \tag{5.95}$$

Using (5.93) with respect to $\mathbf{g}_k^{ada}$, we establish the following bound for $(G_k^{1/2})_i$:

$$\sqrt{k}\cdot\sqrt{(\frac{1}{2}(A^T\mathbf{b})_i))^2+\epsilon} \leq (G_k^{1/2})_i \leq \sqrt{k}\cdot\sqrt{(\frac{3}{2}(A^T\mathbf{b})_i))^2+\epsilon}. \tag{5.96}$$

Taking the inverse of $(G_k^{1/2})_i$, our bounds become:

$$\frac{1}{\sqrt{k}\cdot\sqrt{(\frac{3}{2}(A^T\mathbf{b})_i))^2+\epsilon}} \leq (G_k^{1/2})_i^{-1} \leq \frac{1}{\sqrt{k}\cdot\sqrt{(\frac{1}{2}(A^T\mathbf{b})_i))^2+\epsilon}}. \tag{5.97}$$

We can then bound our update $\Delta\mathbf{x}_k^{ada}(i)$ with respect to $|(A^T\mathbf{b})_i|$:

$$\left|\frac{\eta}{2\sqrt{k}\cdot\sqrt{(\frac{3}{2}(A^T\mathbf{b})_i))^2+\epsilon}}(A^T\mathbf{b})_i\right| \leq \left|\Delta\mathbf{x}_k^{ada}(i)\right| \leq \left|\frac{3\eta}{2\sqrt{k}\cdot\sqrt{(\frac{1}{2}(A^T\mathbf{b})_i))^2+\epsilon}}(A^T\mathbf{b})_i\right| \tag{5.98}$$

Using (5.98) and (5.94) we get the following final bound for our ratio:

$$\left|\frac{3}{\sqrt{k}\cdot\sqrt{(\frac{3}{2}(A^T\mathbf{b})_i)^2+\epsilon}}\right| \le \frac{|\Delta\mathbf{x}_k^{ada}(i)|}{|\Delta\mathbf{x}_k^{gd}(i)|} \le \left|\frac{3}{\sqrt{k}\cdot\sqrt{(\frac{1}{2}(A^T\mathbf{b})_i)^2+\epsilon}}\right|. \tag{5.99}$$

□

Theorem (5.4.3) tells us that when a feature has small or potentially sparse entries, the Adagrad update for that feature will be larger in magnitude compared to GD. Likewise, when the feature has large or non-sparse entries, Adagrad will give a smaller update to that feature compared to GD. In all, the ratio between the update with respect to Adagrad and GD varies inversely with the magnitude of the column $\|A(:,i)\|$ for columns satisfying our assumption. Intuitively, Adagrad will converge to solutions where weights for features that are small in norm, and weights for features that are large in norm are closer together in absolute value, when compared to gradient descent.

5.5 Connection Between ℓ_∞-Norm Solution and Fixed Preconditioned Gradient Descent

In section 5.4, we established the idea of local vs diffuse solutions to the underdetermined system $A\mathbf{x} = \mathbf{b}$ via the introduction of the local-sparsity metric. From here we introduced our experimental findings stating that Adagrad promoted diffuse solutions with respect to the LSM, as well as when compared to gradient descent. We concluded by theoretically showing that Adagrad promoted more diffuse solutions when compared to gradient descent when features are relatively more sparse.

Another key observation that was made from our observations, which are presented in chapter 6, was that the LSM of the minimum ℓ_∞-norm solution and the Adagrad solution share a very similar (low) LSM score. This leads us to question whether the Adagrad preconditioner matrix allows for some sort of connection between the ∞-norm solution and the Adagrad solution. To simplify this question, we, instead, consider the following question: is there a fixed diagonal preconditioner G, such that the ∞-norm solution is equal to the fixed preconditioned gradient descent solution? The idea here is that the ∞-norm solution produces very diffuse solutions compared to the other common p-normed solutions, and Adagrad's solution shares similar diffusion properties, so does the preconditioner matrix somehow steer the training to reach a solution close to the ∞-norm solution?

The following two theorems show that there does exist a construction of G which enables fixed preconditioner GD to converge to the minimum ∞-norm solution. Theorem 5.5.2 shows how, under the right construction of a diagonal weight matrix D, weighted least-squares (WLS) converges to the minimum ∞-norm solution, and theorem 5.5.3 shows that under the correct selection of the fixed preconditioner G (chosen with respect to D), we have that fixed preconditioner GD is equivalent to WLS.

Before we jump in, however, we require the knowledge of a well known result called the Karush-Kuhn-Tucker (KKT) conditions.

Definition 5.5.1 (Relative Interior)**.** Let $S \subseteq \mathbb{R}^d$. The relative interior of S, denoted as relint(S), is the interior of the affine hull (set of all affine combinations of elements in S denoted aff(S)) of S. That is,

$$\operatorname{relint}(S) := \{s \in S \ : \exists \epsilon > 0 \; s.t. \; B(s;\epsilon) \cap \operatorname{aff}(S) \subseteq S\} . \tag{5.100}$$

Definition 5.5.2 (Slater's Condition)**.** For a convex problem:

$$\begin{aligned} &\min_{\mathbf{x}\in\mathbb{R}^d} f_0(\mathbf{x}) \\ s.t. \;\; &f_i(\mathbf{x}) \leq 0, \;\; i = 1,\ldots,m \\ &A\mathbf{x} = b \end{aligned}$$

where $f_0, f_1, \ldots, f_m$ are convex functions, and $D := \cap_{i=1}^m \operatorname{dom}(f_i)$ (convex), Slater's condition states that strong duality holds if there exists a point $\mathbf{x}^* \in \operatorname{relint}(D)$ which is strictly feasible, i.e. satisfies

$$\begin{aligned} &f_i(\mathbf{x}^*) < 0, \;\; i = 1,\ldots,m \\ &A\mathbf{x}^* = b. \end{aligned}$$

Such a point is called a Slater point.

Fact 5.5.1 (KKT Conditions for Convex Problems)**.** *Consider the following general convex problem which is in standard form:*

$$\begin{aligned} &\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x}) \\ s.t. \;\; &h_i(\mathbf{x}) \leq 0, \;\; i = 1,\ldots,m \\ &g_i(\mathbf{x}) = 0, \;\; i = 1,\ldots,r \end{aligned} \tag{5.101}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is convex, h_i are convex for all $i = 1,\ldots,m$, and g_i are affine transformations for all $i = 1,\ldots,r$. Then, if problem (5.101) satisfies Slater's Condition[1]*, then*

[1]Other constraint qualifications such as LICQ and MFCQ may also be considered.

$\mathbf{x}^* \in R^d$ is said to both necessarily and sufficiently be an optimum to the convex problem if it satisfies the following conditions, known as KKT conditions:

$$\begin{aligned}
&\mathbf{0} \in \partial f(\mathbf{x}^*) + \sum_{i=1}^{m} \mu_i \partial h_i(\mathbf{x}^*) + \sum_{j=1}^{r} \lambda_i \partial g_i(\mathbf{x}^*) && \text{(Stationarity)}\\
&\mu_i \cdot h_i(\mathbf{x}^*) = 0, \ \ \forall i && \text{(Complementary Slackness)}\\
&h_i(\mathbf{x}^*) \leq 0, \ g_i(\mathbf{x}^*) = 0, \ \ \forall i, j && \text{(Primal Feasibility)}\\
&\mu_i \geq 0, \ \ \forall i && \text{(Dual Feasibility)}
\end{aligned}$$

for some constants μ_i and $\lambda_i \in \mathbb{R}$ called KKT multipliers.

Theorem 5.5.2. *Let $\mathbf{x}^*_\infty$ be the solution to the minimum ∞-norm problem which we'll denote as $(P1)$ and written as:*

$$\begin{gathered}
(P1)\\
\min_{\mathbf{x}\in\mathbb{R}^d} \|\mathbf{x}\|_\infty\\
s.t \ \ A\mathbf{x} = \mathbf{b}.
\end{gathered}$$

Similarly, let $\mathbf{x}^*_{WLS}$ be the solution to the weighted minimum 2-norm problem which we'll denote as $(P2)$ and written as:

$$\begin{gathered}
(P2)\\
\min_{\mathbf{x}\in\mathbb{R}^d} \frac{1}{2}\|Dx\|_2^2\\
s.t \ \ A\mathbf{x} = \mathbf{b}
\end{gathered}$$

where $D \in \mathbb{R}^{d\times d}$ is a diagonal matrix assigning independent weights to each coordinate of $\mathbf{x}$. Then there exists a D such that $\mathbf{x}^*_\infty = \mathbf{x}^*_{WLS}$.

Proof. In order to show that (P1) and (P2) converge for the same solution for a specific D, we require to construct D such that both $\mathbf{x}^*_{WLS}$ and $\mathbf{x}^*_\infty$ satisfy the KKT conditions of their opposing problem. Let $D \in \mathbb{R}^{d\times d}$ be diagonal. Then the KKT conditions of (P2) are:

$$\begin{aligned}
&A^T\mathbf{z} = D^2\mathbf{x} && \text{(Stationarity)}\\
&A\mathbf{x} = \mathbf{b} && \text{(Primal Feasibility)}
\end{aligned}$$

where $\mathbf{z}$ is the vector of KKT multipliers. Now we'd like to write out the KKT conditions for (P1). Instead, we reframe (P1) to an equivalent linear program (LP) which we'll denote as (P1') and is written as:

$$\begin{aligned}
&(P1') \\
&\min_{t\in\mathbb{R}}(t) \\
s.t\ \ &A\mathbf{x} = \mathbf{b} \\
-&\mathbf{x} + \mathbf{e}t \geq 0 \\
&\mathbf{x} + \mathbf{e}t \geq 0
\end{aligned}$$

where $t > 0$ and $\mathbf{e}$ is the vector of ones with dimension adapted to that of $\mathbf{x}$. Then we have the following KKT conditions for (P1'):

$$\begin{aligned}
&A\mathbf{x} = \mathbf{b} \\
-&\mathbf{e}t \leq \mathbf{x} \leq \mathbf{e}t
\end{aligned} \qquad \text{(Primal Feasibility)}$$

$$\begin{aligned}
&\boldsymbol{\lambda}_i(-\mathbf{x}_i + t) = 0 \ \ \forall i \\
&\boldsymbol{\mu}_i(\mathbf{x}_i + t) = 0 \ \ \forall i
\end{aligned} \qquad \text{(Complementary Slackness)}$$

$$\begin{aligned}
&1 = \mathbf{e}^\top(\boldsymbol{\lambda} - \boldsymbol{\mu}) \\
&A^\top\mathbf{w} = (\boldsymbol{\lambda} - \boldsymbol{\mu})
\end{aligned} \qquad \text{(Stationarity)}$$

$$\begin{aligned}
&\boldsymbol{\lambda} \geq 0 \\
&\boldsymbol{\mu} \geq 0
\end{aligned} \qquad \text{(Dual Feasibility)}$$

where $\mathbf{w}, \boldsymbol{\lambda}$, and $\boldsymbol{\mu}$ are d-dimensional vectors of KKT multipliers. Comparing the KKT conditions of (P1') and (P2), we see that they are the same if

$$D^2\mathbf{x} = (\boldsymbol{\lambda} - \boldsymbol{\mu}). \tag{5.102}$$

We therefore choose the diagonal entries, D_{ii}, of D depending on the following 4 cases of $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$:

Case 1: if $\boldsymbol{\lambda} = \boldsymbol{\mu} = \mathbf{0}$
Then taking $D_{ii} = 0$ we must have that:

$$D_{ii}^2\mathbf{x}_i = \boldsymbol{\lambda}_i - \boldsymbol{\mu}_i.$$

Case 2: if $\boldsymbol{\lambda} > \mathbf{0},\ \boldsymbol{\mu} = \mathbf{0}$
Then we must have that $\mathbf{x}_i = t$ which implies that $\mathbf{x}_i > 0$. Since $\mathbf{x}_i > 0$ we choose $D_{ii} > 0$ satisfying:

$$D_{ii}^2 \mathbf{x}_i = \boldsymbol{\lambda}_i.$$

Case 3: if $\boldsymbol{\lambda} = \mathbf{0},\ \boldsymbol{\mu} > \mathbf{0}$
Then we must have that $\mathbf{x}_i = -t$ which implies that $\mathbf{x}_i < 0$. Since $\mathbf{x}_i < 0$ we choose $D_{ii} > 0$ satisfying:

$$D_{ii}^2 \mathbf{x}_i = -\boldsymbol{\mu}_i.$$

Case 4: if $\boldsymbol{\lambda} \geq \mathbf{0},\ \boldsymbol{\mu} \geq 0$
This scenario would not happen as this will contradict complementary slackness of (P1'). That is, under this case, we would have that $\mathbf{x}_i = t$ and $\mathbf{x}_i = -t$ which is a contradiction.

This concludes the proof. □

So we have shown that it is possible to construct a diagonal weight matrix D such that the solution to (P2) will equal the solution of (P1). The following theorem shows how to select the fixed preconditioner, G, such that fixed preconditioner GD applied to problem (5.3) converges to the same solution as (P2):

Theorem 5.5.3. *Let $A \in \mathbb{R}^{n\times d}$ be underdetermined and full row rank. Let D be the diagonal weight matrix from (P2). If the problem (P1) is such that D can be constructed to be positive definite, set $G = D^{-2}$. Using G as the preconditioner matrix for fixed preconditioned gradient descent under problem (5.3), fixed preconditioner GD will converge to the solution of (P2).*

Proof. First, since D is diagonal, we may rewrite (P2) as

$$\begin{aligned} &\min_{\mathbf{x}\in\mathbb{R}^d} \mathbf{x}^\top D^2 \mathbf{x} \\ &s.t\ \ A\mathbf{x} = \mathbf{b} \end{aligned} \tag{5.103}$$

Letting $\mathbf{z} = D\mathbf{x}$, we get

$$\begin{aligned} &\min_{\mathbf{z}\in\mathbb{R}^d} \mathbf{z}^T\mathbf{z} = \|\mathbf{z}\|_2^2 \\ &s.t \;\; AD^{-1}\mathbf{z} = \mathbf{b} \end{aligned} \tag{5.104}$$

Thus, by the minimum 2-norm solution of least-squares (theorem 5.2.1 when $G = I$), we have that

$$\mathbf{z}^* = (AD^{-1})^T((AD^{-1})(AD^{-1})^T)^{-1}\mathbf{b} \tag{5.105}$$

$$\implies \mathbf{x}^* = D^{-1}(AD^{-1})^T((AD^{-1})(AD^{-1})^T)^{-1}\mathbf{b} \tag{5.106}$$

Letting $D^{-1} = G^{1/2}$ (or rearrange to choose $G = D^{-2}$), we get (5.14) where $X = AG^{1/2}$.

This concludes the proof. $\square$

Theorems 5.5.2 and 5.5.3 together show that for a correct choice of preconditioner, fixed preconditioned gradient descent can achieve the minimum ℓ_∞-norm solution of overparameterized LR.

Additionally, we notice that in the construction of the weight matrix D in the proof of Theorem 5.5.2, to achieve the minimum ∞-norm solution of overparameterized LR, we must have that $D_{ii} = 0$ whenever $|\mathbf{x}_i| < \|\mathbf{x}\|_\infty$[2]. In other words, we have that $D_{ii} = 0$ for all smaller entries (in magnitude) of $\mathbf{x}$. This corresponds to what we witness in Adagrad's behaviour, namely that smaller entries (or features) of $\mathbf{x}$ yield smaller entries of the gradient $\mathbf{g}$, which implies smaller entries of the Adagrad preconditioner, G_{ii}. Recall that in order to achieve the ∞-norm solution, we need the preconditioner to satisfy $G = D^{-2}$ where D is constructed to satisfy Theorem 5.5.2. So if $D_{ii} = 0$ whenever we have small entries of $\mathbf{x}$ (in magnitude), then D_{ii}^{-2} will be infinite and therefore cannot attain a G that will satisfy the ∞-norm solution, but we can get close if the G_{ii} is large which is what is observed for the Adagrad preconditioner.

[2]This follows from primal feasibility and complementary slackness of $(P1')$ which yields case 1 in the proof of Theorem 5.5.2.

5.6 Diffusivity of Adagrad May Not Always Generalize Well

This brief section will cover the connection between our observations of diffusivity of Adagrad's solution in the underdetermined least-squares regime to those posed in the work by Wilson et al. [57]. Recalling from the related-works chapter 2, the work in the paper argues that adaptive optimization methods, like Adagrad, tend to generalize worse than stochastic GD. In their study, the setting of focus is that of over-parameterized binary least-squares classification:

$$\min_{\mathbf{x}\in\mathbb{R}^d} R_S[\mathbf{x}] := \frac{1}{2}\|A\mathbf{x}-\mathbf{b}\|_2^2 \tag{5.107}$$

where $A \in \mathbb{R}^{n\times d}$ $(n < d)$, and $\mathbf{b} \in \mathbb{R}^n$ is a vector of labels in $\{-1, 1\}$. We notice that this is a similar regime of study to our own except that they limit the space of the solution vector $\mathbf{b} \in \{-1, 1\}$. As was discussed previously, (stochastic) GD under this regime will converge to the minimum 2-norm solution $\mathbf{x}^{SGD} = A^T(AA^T)^{-1}\mathbf{b}$, which is considered as the solution with largest margin out of all solutions of the form: $A\mathbf{x} = \mathbf{b}$ when speaking of the task of classification.

In terms of adaptive methods, the authors propose the following lemma which captures the solution form of general adaptive methods of the form:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k H_k^{-1}\tilde{\nabla} f(\mathbf{x}_k + \gamma_k(\mathbf{x}_k - \mathbf{x}_k - 1)) + \beta_k H_k^{-1} H_{k-1}(\mathbf{x}_k - \mathbf{x}_{k-1}) \tag{5.108}$$

where H_k is a positive definite preconditioner matrix and $\tilde{\nabla}$ denotes the use of either a full or stochastic gradient. For the case of Adagrad, setting H_k to be (4.12), $\alpha_k \equiv \alpha > 0$, $\beta_k = 0$, and $\gamma_k = 0$ in equation (5.108) we retrieve the Adagrad update. Denote $sign(\mathbf{x})$ to be the function which maps each component of its input to its sign. The following holds for diagonal H_k's:

Lemma 5.6.1. *Suppose there exists a scalar $c \in \mathbb{R}$ such that $A\ sign(A^T\mathbf{b}) = c\mathbf{b}$. Then, when initialized at $\mathbf{x}_0 = \mathbf{0}$, Adagrad, Adam, and RMSProp all converge to the unique solution $\mathbf{x} \propto sign(A^T\mathbf{b})$.*

Proof. See [57]. □

What this lemma is concluding at is that, if there exists a solution of $A\mathbf{x} = \mathbf{b}$ that is proportional to $\text{sign}(A^T\mathbf{b})$, then Adagrad (as well as the other adaptive methods mentioned) will converge to said value. In other words, the solution generated by Adagrad

$\mathbf{x}^{ada}$, will have coordinates of the form $\pm\tau$ for some positive constant τ. Furthermore, the authors construct a particular example demonstrating how Adagrad will classify all unseen data as the positive class [57, Section 3.3]. As we'll discuss in a moment, the construction of the problem is such that solutions which are locally sparse will perform better than those which are diffuse.

This behaviour aligns with our findings in sections 5.4 and 5.5. First, we saw that Adagrad promoted diffuse solutions to the over-parameterized least-squares regime. By Lemma 5.6.1 we have that solutions of Adagrad, $\mathbf{x}^{ada}$, under this particular case of binary least-squares classification, will yield solutions in which the coordinates of $\mathbf{x}^{ada}$ will have equal magnitude. The LSM of such a solution will yield us a value which reaches the lower bound of the LSM (refer to (5.57)). That is, we should expect that Adagrad produces diffuse solutions in this special case regardless of what we set the step-size α to.

Additionally, the converged solutions produced by Adagrad in this particular case yield low ℓ_∞-norm values, more so than when compared to the solution's ℓ_2-norm [57]. This connects to what we spoke about in section 5.5 and what we will see in our experiments in section 6.1 in that there exists similarities between the solutions produced by Adagrad and the minimum infinity-norm solution. As such, training using optimizers that yield diffuse solutions on cases in which a locally sparse solution is needed in order to generalize well will cause Adagrad's generalization performance to suffer. The particular example constructed in [57, Section 3.3] benefits optimizers that select sparser solutions as only the first entry of each sample is needed to perfectly classify the sample; a diffuse solution will inevitably distribute the weight given to its parameters more evenly.

The question then becomes, will solutions with low infinity norm (or in our terminology, low LSM) always generalize poorly? In section 6.2, we demonstrate that one may also construct certain problems of over-parameterized least-squares in which these diffuse solutions generalize better than those solutions with low ℓ_2-norm. This contrast lends its hand to why, in practice, adaptive methods sometimes still out-perform the non-adaptive methods in certain deep learning regimes. This leads to the train of thought that adaptive methods are not universal optimizers, and that one should not blindly select an optimizer when training their models. It also highlights the importance of understanding different optimizers' solution biases.

Chapter 6

Experimental Results

In this chapter we present and discuss our experimental findings based off what was discussed in chapter 5. Specifically, we first show empirical evidence of Adagrad promoting more diffuse solutions when compared to the least squares minimization scenario (i.e. gradient descent). From this, we present a case where such solutions chosen by Adagrad are advantageous, namely in the interpolation of a line using bump functions.

6.1 Adagrad Promotes Diffuse Solutions

In chapter 5 we introduced the local-sparsity metric as a way of measuring the approximate sparsity of a solution generated by an algorithm. In short, the output of the LSM is bounded as in (5.52) where the output being closer to the lower bound implies the input is diffuse or not approximately sparse, while the output being closer to the upper bound implies the input is approximately sparse in the sense that certain coordinates dominate the rest. From such a metric, we should expect that the minimum ∞-norm solution of overparameterized LR, which promotes solutions in which the coordinates are approximately equal, and the minimum one-norm solution which promotes sparse solutions should be on opposite ends of the bound's spectrum. Denoting these as the infinity-norm and one-norm solutions respectively, we expect that the infinity-norm solution lies very close to the lower bound of the LSM, while the one-norm lies close to the upper bound of the LSM. As such, we can also think of the two-norm solution as being somewhere in between. As a quick recall, we denote the minimum α-norm solution to be the solution outputted by problem (5.50) where the norm on $\mathbf{x}$ is chosen to be α.

The tables that follow contain the average LSM-score computed on the outputs of selected optimizers over five runs. Each column represents the runs performed using the indicated step-size $\eta = [0.001, 0.01, 0.1, 1]$, where a single run, used to calculate the average, consists of solving (5.3) where we randomly generate an underdetermined matrix, A, and random solution b. The runs computed are independent across tables. Each row represents the LSM-scores generated by the two-norm, one-norm, infinity-norm, Adagrad, and stochastic Adagrad, respectively. Each table generates the data based off of a fixed number of observations n, and features d, under the rule $n \approx \sqrt{d}$. Figure 6.1 contains the table such that $n = 5$ and $d = 40$. Figure 6.2 contains the table such that $n = 20$ and $d = 400$, and Figure 6.3 contains the table such that $n = 63$ and $d = 4000$.

Between each table: 6.1, 6.2, and 6.3, the step-sizes η that vary between the tables only affects the performance of both Adagrad and stochastic Adagrad as these are the only two iterative algorithms programmed. The α-norm solutions ($\alpha \in \{1, 2, \infty\}$) are generated using the Matlab convex programming solver *CVX* which does not utilise η. We also note that we do not include gradient descent as an optimizer since GD converges to the 2-Norm solution as was seen in chapter 5.

Following these tables are four bar graphs (Figure 6.1) of the LSM scores computed in Table 6.3. Each bar graph captures these averaged LSM scores with respect to one of the four step-sizes $\eta = [0.001, 0.01, 0.1, 1]$. Figure 6.2 captures the same data but focusing just on Adagrad and stochastic Adagrad as they are the two optimizers that are affected by η.

* We note that stochastic Adagrad is trained with a step-size of η/n where n is the number of observations (or rows) in the data matrix A since each step only considers a single observation.

Observations $n = 5$; **Features** $d = 40$

	$\eta = 0.001$	$\eta = 0.01$	$\eta = 0.1$	$\eta = 1$
2-Norm	0.19832	0.19865	0.19770	0.19317
1-Norm	0.53652	0.59118	0.52983	0.50106
∞-Norm	0.16254	0.16195	0.16151	0.16210
Ada.	0.16352	0.16410	0.17927	0.28220
S-Ada.*	0.18383	0.18307	0.18727	0.20697

Table 6.1: Table consisting of the average of 5 runs across the 5 optimizers. Each column represent the runs with step-size chosen from $\eta = [0.001, 0.01, 0.1, 1]$ respectively, and each row represents the results produced by ℓ_2-norm, ℓ_1-norm, ℓ_∞-norm, Adagrad, and stochastic Adagrad, respectively. Each run used to calculate the average generates a new random data matrix A and target $\mathbf{b}$ such that $n = 5$, $d = 40$. The lower bound for the LSM ≈ 0.1581.

Observations $n = 20$; **Features** $d = 400$

	$\eta = 0.001$	$\eta = 0.01$	$\eta = 0.1$	$\eta = 1$
2-Norm	0.06241	0.06209	0.06213	0.06165
1-Norm	0.29995	0.30501	0.29326	0.28905
∞-Norm	0.05033	0.05037	0.05036	0.05039
Ada.	0.05126	0.05177	0.07989	0.09937
S-Ada.*	0.06096	0.06085	0.06110	0.06308

Table 6.2: Table consisting of the average of 5 runs across the 5 optimizers. Each column represent the runs with step-size chosen from $\eta = [0.001, 0.01, 0.1, 1]$ respectively, and each row represents the results produced by ℓ_2-norm, ℓ_1-norm, ℓ_∞-norm, Adagrad, and stochastic Adagrad, respectively. Each run used to calculate the average generates a new random data matrix A and target $\mathbf{b}$ such that $n = 20$, $d = 400$. The lower bound for the LSM ≈ 0.05.

Observations $n = 63$; **Features** $d = 4000$

	$\eta = 0.001$	$\eta = 0.01$	$\eta = 0.1$	$\eta = 1$
2-Norm	0.01980	0.01983	0.01979	0.01982
1-Norm	0.16770	0.16908	0.17042	0.17876
∞-Norm	0.01585	0.01586	0.01585	0.01586
Ada.	0.01608	0.02095	0.03688	0.03820
S-Ada.*	0.01959	0.01963	0.01997	0.02027

Table 6.3: Table consisting of the average of 5 runs across the 5 optimizers. Each column represent the runs with step-size chosen from $\eta = [0.001, 0.01, 0.1, 1]$ respectively, and each row represents the results produced by ℓ_2-norm, ℓ_1-norm, ℓ_∞-norm, Adagrad, and stochastic Adagrad, respectively. Each run used to calculate the average generates a new random data matrix A and target $\mathbf{b}$ such that $n = 63$, $d = 4000$. The lower bound for the LSM ≈ 0.01581.

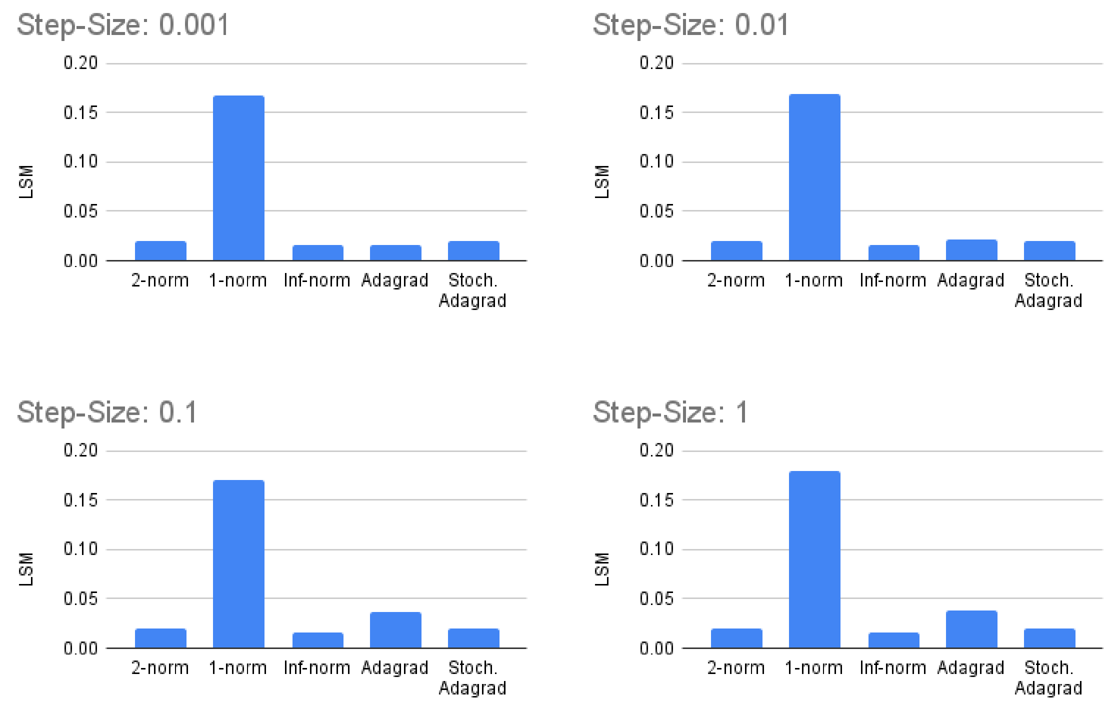
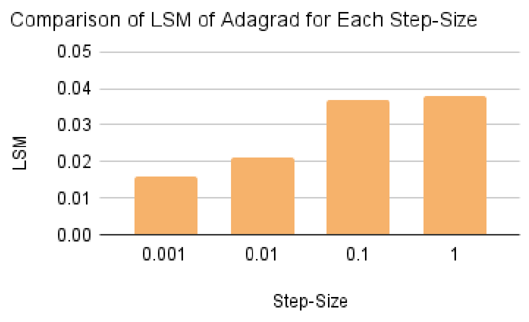


Figure 6.1: Bar charts of the LSM scores of Table 6.3. Each chart presents the data of one column of the table which represents the LSM result of each optimizer for the given step-sizes $\eta \in [0.001, 0.01, 0.1, 1]$.



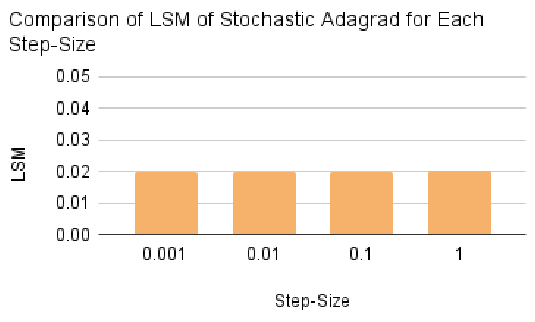


Figure 6.2: Bar Charts combining the LSM scores of **(left)** Adagrad, and **(right)** Stochastic Adagrad from Figure 6.1. Each bar indicates the LSM score for a step-size $\eta \in [0.001, 0.01, 0.1, 1]$. This data is deduced from Table 6.3.

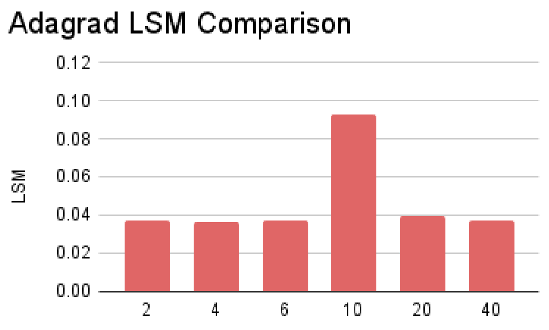
It is evident that, across the tables 6.1, 6.2, and 6.3, the same pattern occurs as the step size varies; That is, the size of the data does not seem to have an affect on the nature of the diffusion pattern; It only decreases the lower bound attainable by the LSM. What we do notice, however, is that the step-size η does have an affect on Adagrad's LSM score. Namely, as step size decreases, the Adagrad LSM score decreases. This is telling us that, for small enough step size, Adagrad promotes non approximately sparse solutions, but for large enough step size, the behaviour no longer holds and, in fact, promotes approximately sparser solutions to that compared with the two-norm solution.

We also confirm what was discussed at the beginning of this section in that the ∞-norm stays very close to the lower bound while the 1-norm has a very high score when compared to the other optimizers. To understand this intuitively, we may think of the geometry that each norm induces when considering the set of vectors $\mathbf{x}$ s.t $\|\mathbf{x}\| = 1$ in $\mathbb{R}^2$. For the 1-norm, we have a diamond centered at the origin where each vertex is located on $(1, 1), (0, 1), (-1, 1), (0, -1)$. Therefore, when attempting to find the 1-norm solution, these vertices will be the first to intersect the hyperplane of solutions generated by the underdetermined system $A\mathbf{x} = \mathbf{b}$. So, intuitively, the 1-norm solution will promote sparse solutions and therefore have a high LSM score. A similar idea is applied to the ∞-norm, only now we note that the ∞-norm is a square centered at zero with each vertex located at $(1, 1), (-1, 1), (-1, -1), (1, -1)$. So for the ∞-norm solution, these vertices will likely be the first to intersect with the hyperplane of solutions and we see that these vertices are diffuse giving us a low LSM score.

Moreover, when comparing the LSM score of Adagrad to the LSM score of the 2-norm (or, in other words, the LSM score of GD), we see that LSM-Adagrad is less than LSM-

2-Norm when η is sufficiently small, but then no longer holds when η is sufficiently large. This observation matches up with Theorem (5.4.3) in that, in order for the bound on the ratio (5.63) to hold, we require η small enough. Intuitively, the preconditioner matrix of Adagrad suppresses the growth of strong signaled features while promoting the growth of low signaled features. When equipped with a small step-size η, this suppresses the strong signaled features even more while only dampening the promotion of low signaled features. With such restriction on the strongly signaled features and net promotion on the low signaled features, we can imagine that early iterations of Adagrad shift the direction of descent towards the low signaled features enough to converge to a more diffuse solution.

Turning to stochastic Adagrad, there does seem to be a slight increase in LSM as we vary η towards 1, however it is hard to tell if it is noise or not. To ensure that this is not just noise, we perform additional runs with higher step-sizes. The following two bar graph shows the average LSM score of stochastic Adagrad, and Adagrad for comparison, over 5 runs when $n = 63$, $d = 4000$, and we vary $\eta = [2, 4, 6, 10, 20, 40]$. We chose these dimensions in order to remain consistent with Figure 6.2.



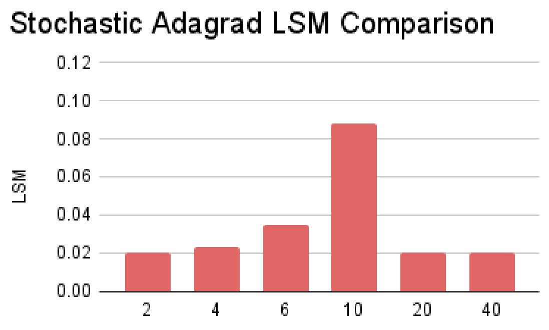


Figure 6.3: Bar charts containing additional averaged LSM score runs for a large magnitude of varying step-sizes η, for **(Left)** Adagrad, and **(right)** stochastic Adagrad. Runs were performed on generated data with $n = 20$, $d = 400$.

In the case for Adagrad, we see that, except for the anomaly at $\eta = 10$, the LSM seems to stabilize after $\eta = 1$. For stochastic Adagrad we see some additional growth towards the LSM of full-gradient Adagrad at around step-size $\eta = 6$. The same anomaly occurs at $\eta = 10$ and then the step-size drops back to the more lower LSM scores realized. We thus see that stochastic Adagrad does grow its LSM but required a larger step-size than full-gradient Adagrad.

6.2 A Case For Diffuse Solutions: Interpolation

In this section, we present our empirical findings for an application where having a low LSM score has an advantage. Such a case is the problem of interpolating a function using bump-like functions. The inspiration behind this problem comes from radial basis function (RBF) interpolation which was briefly discussed in the background section 3.4. Recall that in RBF interpolation, we are attempting to find optimal weights $\mathbf{x}$ such that $s(\zeta_i;\mathbf{x}) = f(\zeta_i)$ for $i = 1,\ldots,N$ for some, possibly unknown, function f. We concluded that this was equivalent to solving a linear regression problem $A\mathbf{x} = \mathbf{b}$ where A is a matrix with entries satisfying the form $a_{ij} = \psi(\|\zeta_j - \zeta_i\|)$, and the vector $\mathbf{b}$ is such that $\mathbf{b}_i = f(\zeta_i)$.

In this experiment, we aim to interpolate a line in $\mathbb{R}^2$ using a system similar to RBF interpolation. That is, we attempt to solve for weights $\mathbf{x}$ such that

$$s(\zeta_i;\mathbf{x}) = \sum_{k=1}^{N} \mathbf{x}_i \cdot \psi(\|\zeta_i - \zeta_k\|_2) = f(\zeta_i) \;\; \forall i = 1,\ldots,N \tag{6.1}$$

except we don't use a common RBF. Instead, we consider the following spike-like functions:

$$\phi\left(\|r-\zeta\|_2\right) := \max\left\{0, \frac{\alpha - \|r-\zeta\|_2}{\alpha^2}\right\} \tag{6.2}$$

where $\zeta \in \mathbb{R}^m$, $m > 0$, is a centre, and $\alpha \in \mathbb{R}$ is a shape parameter which affects both the width and height of the spike. This function is constructed such that the integral evaluates to 1. Figure 6.4 shows an example when $\alpha = 0.39$ and $\zeta = 0.87$.

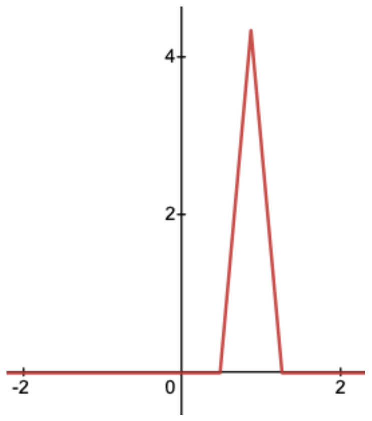


Figure 6.4: Spike function generated by (6.2) when $\alpha = 0.39$, and the centre $\zeta = 0.87$.

Our experiment set up is therefore the following: we'd like to consider the scenario where we randomly generate a data set $A \in \mathbb{R}^{2n \times d}$ where $2n < d$ and labels $\mathbf{b} \in \mathbb{R}^{2n}$ such that the entries of A are defined by

$$a_{ij} = \max\left\{0, \frac{\alpha_j - \|r_i - \zeta_j\|_2}{\alpha_j^2}\right\} \tag{6.3}$$

for all $i = 1, \ldots, 2n$ and $j = 1, \ldots d$; where $r_i \in \mathbb{R}$ are randomly generated using a (0,1)-uniform distribution, $i = 1, \ldots, 2n$, α_j are randomly generated using a (0,1)-uniform distribution scaled by 0.1, $j = 1, \ldots, d$, and the centres ζ_j are randomly generated using a (0,1)-uniform distribution, $j = 1, \ldots, d$. The coordinates of $\mathbf{b}$ are taken to be the r_i, that is, $\mathbf{b}_i = r_i$, for $i = 1, \ldots, 2n$.

From here, we solve the underdetermined system $\bar{A}\mathbf{x} = \bar{\mathbf{b}}$ where $\bar{A} \in \mathbb{R}^{n \times d}$ and $\bar{\mathbf{b}} \in \mathbb{R}^n$ are taken to be the reduced data-label pair used for training. The goal is to see how well each optimizers' solution generalizes to the entire data, i.e. how well does each optimizers' solution do in interpolating a linear line in $\mathbb{R}^2$. We measure how well an optimzers' solution generalizes to the entire data set by comparing their residuals:

$$\ell(\mathbf{x}^*) = \|A\mathbf{x}^* - \mathbf{b}\|_2. \tag{6.4}$$

Since each run randomly generates a new line and points to interpolate, the results found below are normalized with respect to the ℓ_2 norm of $\mathbf{b}$. Ideally, under such spike functions, we should expect solutions that are non-approximately sparse to perform better since using more spike functions allows us to get a better interpolation.

In the following tables, we compare the residuals that each optimizers' solution (trained on a partitioned training set) outputs when ran on the entire data set. In this experiment, we only vary the number of observations n per group of tables, and within each group we vary the step-size $\eta = [0.001, 0.01, 0.1, 1]$. Varying d does not seem to have any noticeable effects. The optimizers used are the 2-norm, 1-norm, ∞-norm, Adagrad, and Stochastic Adagrad.

Following these tables are visual plots of the performance of each model on a select number of runs. Namely, the plots associate with data from the columns with step-sizes $\eta = \{0.001, 1\}$ of Table 6.4, and of Table 6.6. These were selected to visually show the two ends of the spectrum, namely, smallest step size vs. largest step size, and smallest data set vs. largest data set.

** We note that stochastic Adagrad is trained with a step-size of η/n where n is the number of observations (or rows) in the data matrix A since each step only considers a single observation.

Observations $2n = 80$; **Features** $d = 2000$

	$\eta = 0.001$	$\eta = 0.01$	$\eta = 0.1$	$\eta = 1$
2-Norm	0.24751	0.23289	0.25968	0.24879
1-Norm	0.56325	0.53794	0.57983	0.55057
∞-Norm	0.15973	0.17003	0.16318	0.18108
Ada.	0.10292	0.11475	0.10378	0.33027
S-Ada.**	0.09711	0.09249	0.24902	2.20948

Table 6.4: Each column is the average normalized full-data (both in and out-of-sample) residual of the 2-norm, 1-norm, infinity-norm, Adagrad, and stochastic Adagrad optimizers for a particular step-size $\eta = [0.001, 0.01, 0.1, 1]$ over 5 runs. The full data set is of shape $(A, \mathbf{b}) \in \mathbb{R}^{80\times 2000} \times \mathbb{R}^{80}$.

Observations $2n = 160$; **Features** $d = 2000$

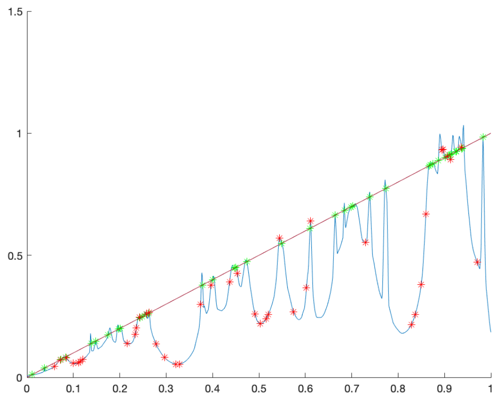
	$\eta = 0.001$	$\eta = 0.01$	$\eta = 0.1$	$\eta = 1$
2-Norm	0.11218	0.10684	0.17305	0.12093
1-Norm	0.42642	0.39539	0.48096	0.42496
∞-Norm	0.08294	0.08363	0.12303	0.08865
Ada.	0.05502	0.05632	0.21842	0.11038
S-Ada.**	0.05002	0.04421	0.11949	0.86796

Table 6.5: Each column is the average normalized full-data (both in and out-of-sample) residual of the 2-norm, 1-norm, infinity-norm, Adagrad, and stochastic Adagrad optimizers for a particular step-size $\eta = [0.001, 0.01, 0.1, 1]$ over 5 runs. The full data set is of shape $(A, \mathbf{b}) \in \mathbb{R}^{160\times 2000} \times \mathbb{R}^{160}$.

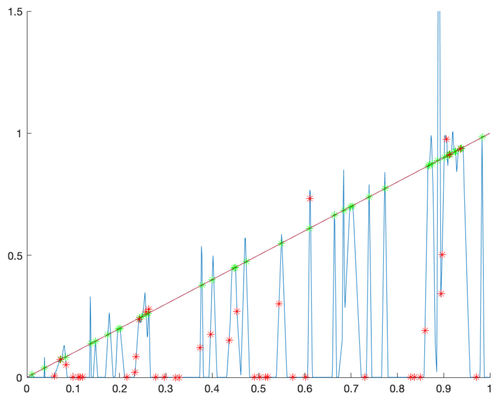
Observations $2n = 200$; **Features** $d = 2000$

	$\eta = 0.001$	$\eta = 0.01$	$\eta = 0.1$	$\eta = 1$
2-Norm	0.08696	0.07428	0.1099	0.09000
1-Norm	0.32866	0.34824	0.42222	0.40572
∞-Norm	0.07141	0.05749	0.08477	0.06877
Ada.	0.05099	0.04406	0.07445	0.05695
S-Ada.**	0.07531	0.02878	0.07157	0.59742

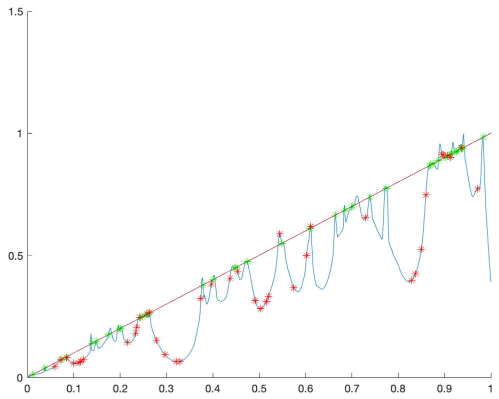
Table 6.6: Each column is the average normalized full-data (both in and out-of-sample) residual of the 2-norm, 1-norm, infinity-norm, Adagrad, and stochastic Adagrad optimizers for a particular step-size $\eta = [0.001, 0.01, 0.1, 1]$ over 5 runs. The full data set is of shape $(A, \mathbf{b}) \in \mathbb{R}^{200\times 2000} \times \mathbb{R}^{200}$.



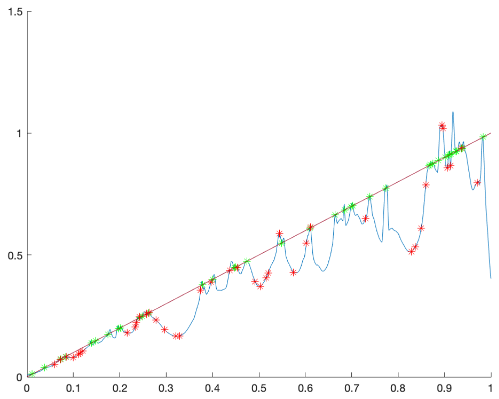
(a) ℓ_2 Norm



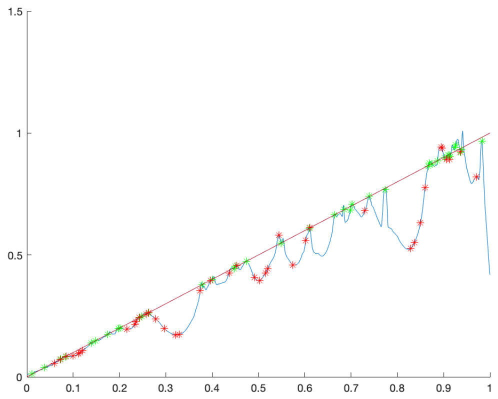
(b) ℓ_1 Norm



(c) ℓ_∞ Norm

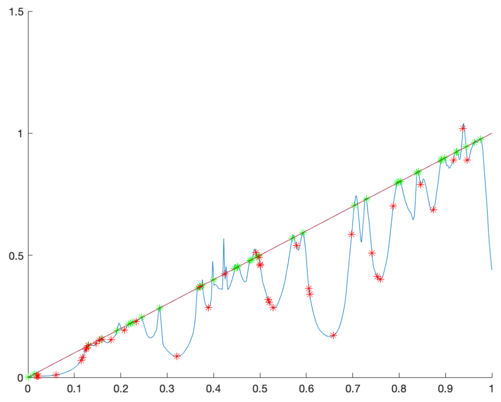


(d) Adagrad

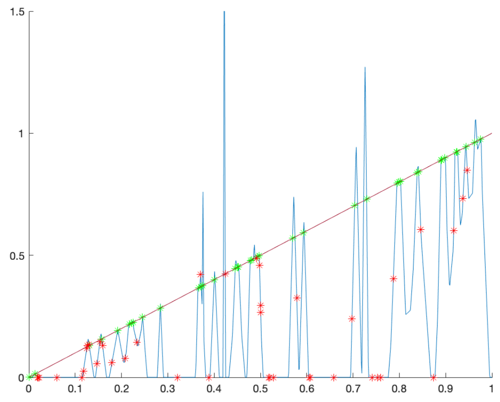


(e) Stochastic Adagrad

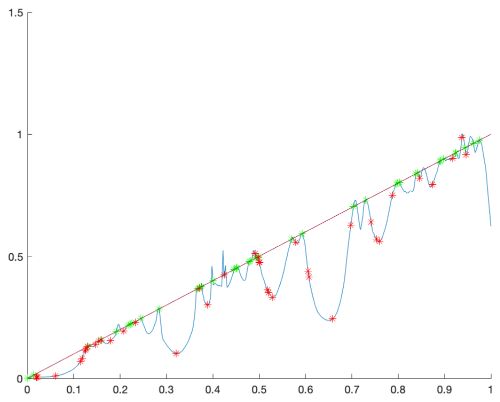
Figure 6.5: Visualized plot of **(a)** 2-norm, **(b)** 1-norm, **(c)** ∞-norm, **(d)** Adagrad, and **(e)** Stochastic Adagrad interpolated line on full data set corresponding to Table 6.4, $\eta = 0.001$. All points present on the graphs are the interpolated points of the line by the respective model. The green points are the interpolated points in which the model had access to the true values for training. The red points are the interpolated points at the unseen data. Together, both green and red points make up all the data. The red line is the true function, $y = x$ that is trying to be interpolated.



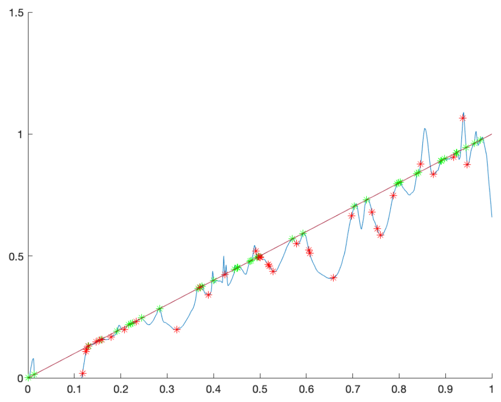
(a) ℓ_2 Norm



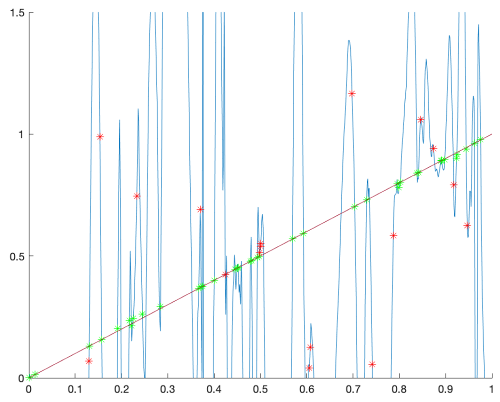
(b) ℓ_1 Norm



(c) ℓ_∞ Norm

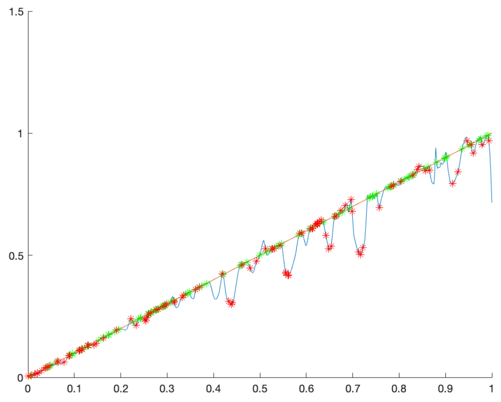


(d) Adagrad

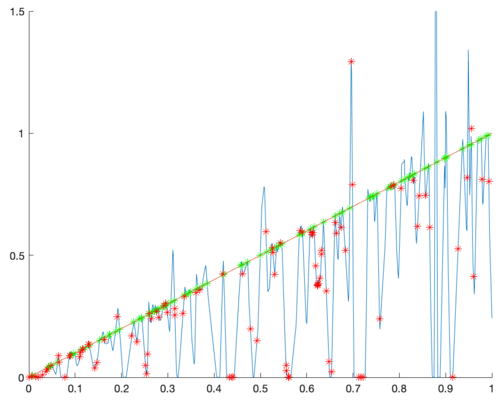


(e) Stochastic Adagrad

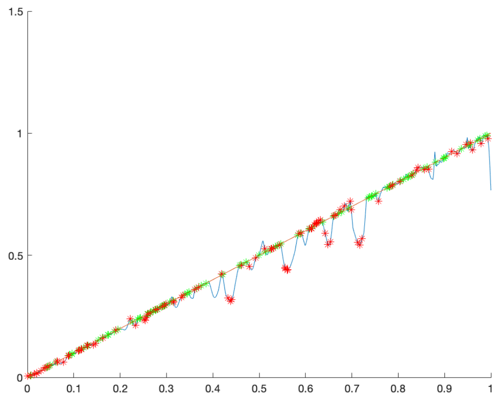
Figure 6.6: Visualized plot of **(a)** 2-norm, **(b)** 1-norm, **(c)** ∞-norm, **(d)** Adagrad, and **(e)** Stochastic Adagrad interpolated line on full data set corresponding to Table 6.4, $\eta = 1$. All points present on the graphs are the interpolated points of the line by the respective model. The green points are the interpolated points in which the model had access to the true values for training. The red points are the interpolated points at the unseen data. Together, both green and red points make up all the data. The red line is the true function, $y = x$ that is trying to be interpolated.



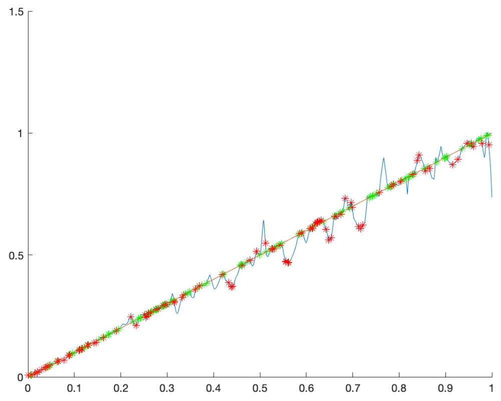
(a) ℓ_2 Norm



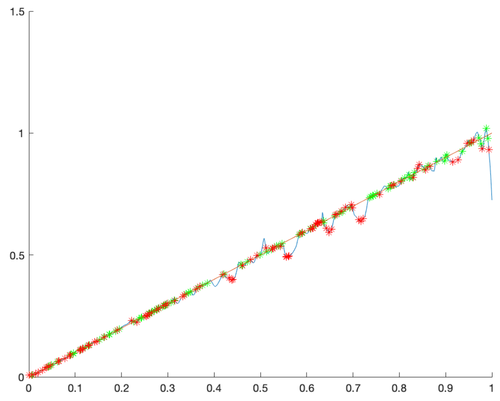
(b) ℓ_1 Norm



(c) ℓ_∞ Norm

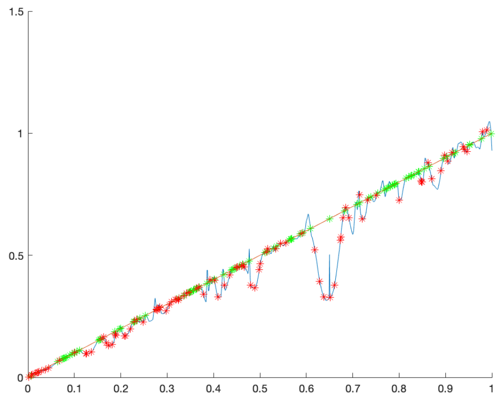


(d) Adagrad

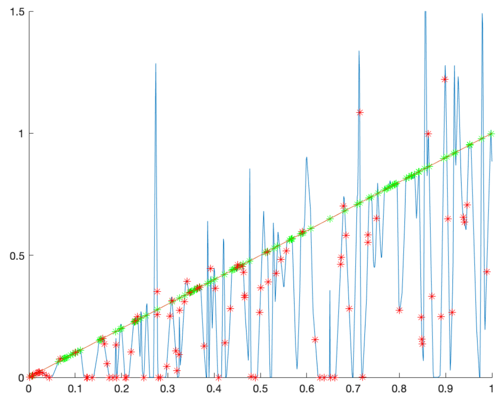


(e) Stochastic Adagrad

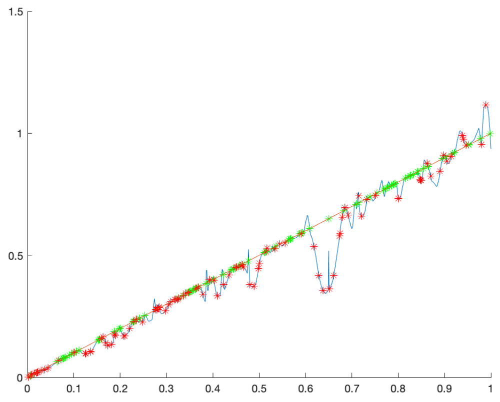
Figure 6.7: Visualized plot of **(a)** 2-norm, **(b)** 1-norm, **(c)** ∞-norm, **(d)** Adagrad, and **(e)** Stochastic Adagrad interpolated line on full data set corresponding to Table 6.6, $\eta = 0.001$. All points present on the graphs are the interpolated points of the line by the respective model. The green points are the interpolated points in which the model had access to the true values for training. The red points are the interpolated points at the unseen data. Together, both green and red points make up all the data. The red line is the true function, $y = x$ that is trying to be interpolated.



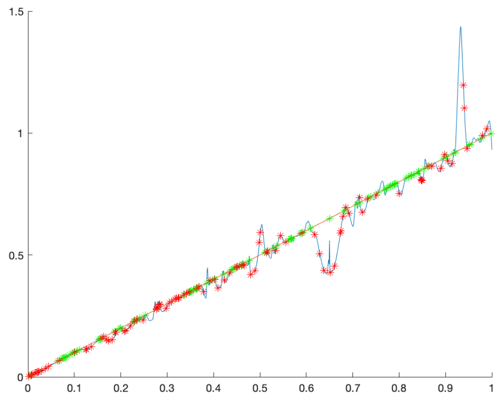
(a) ℓ_2 Norm



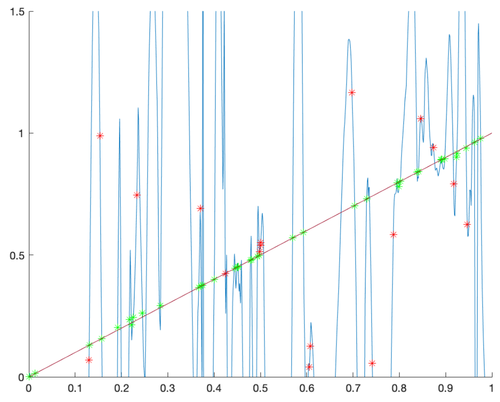
(b) ℓ_1 Norm



(c) ℓ_∞ Norm



(d) Adagrad



(e) Stochastic Adagrad

Figure 6.8: Visualized plot of **(a)** 2-norm, **(b)** 1-norm, **(c)** ∞-norm, **(d)** Adagrad, and **(e)** Stochastic Adagrad interpolated line on full data set corresponding to Table 6.6, $\eta = 1$. All points present on the graphs are the interpolated points of the line by the respective model. The green points are the interpolated points in which the model had access to the true values for training. The red points are the interpolated points at the unseen data. Together, both green and red points make up all the data. The red line is the true function, $y = x$ that is trying to be interpolated.

We notice that our intuition that more diffuse solutions will perform better holds. This can be seen by comparing the 2-norm, 1-norm, and ∞-norm. We saw from our previous experiments that the 1-norm and ∞-norm sit at opposite ends of the spectrum when it comes to the LSM, while the 2-norm hangs somewhere in between, usually closer to the ∞-norm score. Our intuition told us that the 1-norm and 2-norm should perform worse than the ∞-norm because both the 1 and 2-norms promote more approximately sparse solutions. When this happens, a lot of the weights set to the different spike functions have a higher chance of being zero (or local to a select few weights) when compared to the ∞-norm. This prevents us from being able to use more of the spike functions leading to only a handful of spikes having to compensate. Ideally, since we are interpolating using very "ugly" functions, using a larger number of them can enable better interpolation performance. The resulting interpolated line is much more jagged for the 1-norm (and somewhat for the 2-norm) when compared to the ∞-norm; see, for example, Figure 6.5 **(a), (b), (c)**. This is also confirmed with our residual data found in the Tables 6.4, 6.5 and 6.6. We see that, when comparing the three norms (ℓ_1, ℓ_2, ℓ_∞), the ∞-norm has the lowest residual across each table, and, as well, we see that by increasing the training sizes (e.g. results from Table 6.4 vs. Table 6.6), the more spike functions we provide, the better each optimizer performs. This lends its hand to both intuitive ideas expressed above, namely, lower LSM scoring optimizers have lower residuals, and more spike functions allow for a better interpolation.

Bringing Adagrad and its stochastic variant into the picture, we see that it, as well, satisfies these intuitions, however, the residual results of Adagrad allow us to realize something additional to the above intuition. From the previous experiments involving the LSM scores, we saw that, on average, the LSM of Adagrad was on par with the LSM of the ∞-norm solution. Yet, we see that the residuals of Adagrad (and stochastic Adagrad when step size is low) out performs the ∞-norm solution, i.e. are lower. Does this mean lower LSM scoring is not the only factor in the picture? One possibility is that, under this construction of A, Adagrad is getting a lower LSM score than the ∞-norm. This intuition seems to be holding experimentally by observing Table 6.7 1 and 2. We see that, in these tables, Adagrad has the lower average LSM score and a lower residual score when compared to the ∞-norm. We note that in Table 6.7 table 1, that stochastic Adagrad has a slightly better LSM than Adagrad but slightly worse average residual. However, the difference in LSM between Adagrad and its stochastic variant, here, is very small and can very likely be noise.

The next thing that jumps out at us from the Tables 6.4, 6.5, and 6.6, is that, unlike the Tables 6.1, 6.2, and 6.3, increasing the step size does not seem to be affecting the residual of Adagrad. If residual is correlated to the LSM score, then our

Table 1: $2n = 80$, $d = 2000$, $\eta = 0.001$

	Avg. LSM	Avg. Residual
2-Norm	0.03119	0.26298
1-Norm	0.21681	0.59386
∞-Norm	0.03036	0.19285
Ada.	0.0288	0.12773
S-Ada.**	0.02842	0.1737

Table 2: $2n = 80$, $d = 2000$, $\eta = 1$

	Avg. LSM	Avg. Residual
2-Norm	0.03067	0.26447
1-Norm	0.22333	0.55092
∞-Norm	0.02913	0.17707
Ada.	0.02813	0.13734
S-Ada.**	0.03106	2.48108

Table 3: $2n = 80$, $d = 2000$, $\eta = 10$

	Avg. LSM	Avg. Residual
2-Norm	0.03131	0.22139
1-Norm	0.22045	0.5679
∞-Norm	0.03014	0.16115
Ada.	0.20481	0.57744
S-Ada.**	0.03138	20.71838

Table 6.7: Tables consisting of the averages over 5 runs of the LSM score and normalized residuals for the 2-norm, 1-norm, ∞-norm, Adagrad, and stochastic Adagrad. Each table differs by the step size used. Runs across tables are independent.

previous experiment tells us that we should expect the residual of Adagrad to increase as the step-size η increases. This seems to be holding for stochastic Adagrad, but not for full gradient Adagrad. It turns out that the behaviour is still holding. To see this, observe the tables in Table 6.7, we see that the behaviour is present but we needed our step-size to be much larger in this scenario. Why is this the case? Recall that, by Theorem 5.4.3, under the assumption that a column of A satisfies (5.62), then up to some iteration k that depends inversely on $\|A\|_2$, Adagrad will satisfy a larger growth for sparser features when compared to the 2-norm. In our construction of A, it turns out that many features are very sparse, so $\|A\|_2$ will be very small causing the number of iteration of k to be quite large before our effect no longer holds. So it is possible that convergence is reached before such a k takes effect. Likewise, since $\|A\|_2$ is very small, we also have, by Theorem (5.4.3), that our choice of η can be very large. So indeed, nothing is breaking down here, just that our upper bounds for both k and η, such that the effects of Theorem 5.4.3 holds, for this class of problem are larger than the problem considered in Tables 6.1, 6.2, and 6.3.

Referring back to Table 6.7, we still notice that, the average LSM of stochastic Adagrad stays close to the LSM of the ∞-norm, but its ability to generalize becomes worse when step size increases. It is possible that the optimizer is failing to train. The following plots visualize the training and full-data errors with respect to the error $\|A\mathbf{x}_k - \mathbf{b}\|_2$:

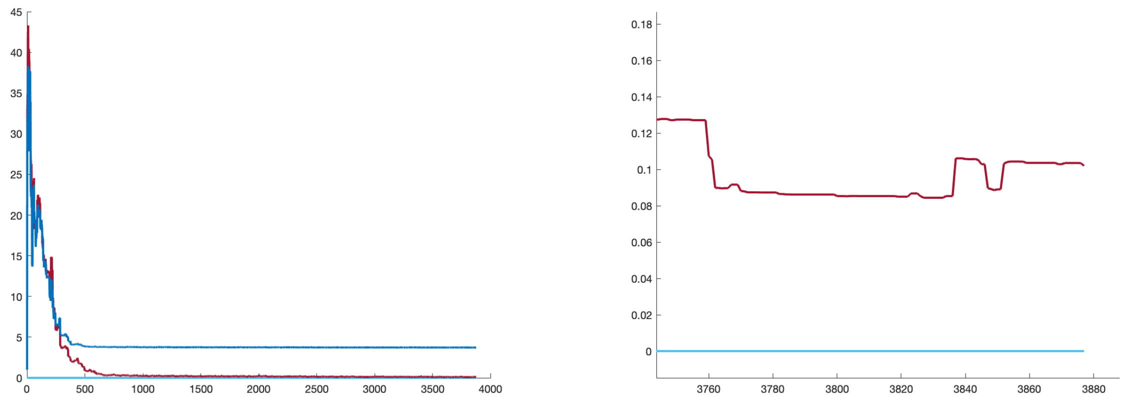


Figure 6.9: **(left)** Error plot of **(Dark Blue)** full-data, and **(red)** training-data of stochastic Adagrad on the interpolation problem with $\eta = 1$, $2n = 80$, and $d = 2000$ normalized with respect to the target; **(right)** zoom-in of left plot showing training error compared to zero error at the final iterations.

From Figure 6.9, it is evident that the training stagnates roughly around a training error of 0.1. It is possible that, due to the large step-size, stochastic Adagrad is jumping around a neighborhood of the solution. We saw in section 3.2.2 that a decaying step-size is required in order to converge. As we increase the step-size, the neighborhood of the solution stochastic Adagrad bounces around gets larger which in turn provides a worse training error, and therefore, a worse generalization error.

Chapter 7

Conclusion

In this thesis, we introduced a class of first order methods known as adaptive gradient methods with the intent on studying their algorithmic biases in the underdetermined linear regression regime. Focusing particularly on the Adagrad algorithm, we first analyzed the simpler case of fixed preconditioned gradient descent. We showed convergence in the underdetermined linear regression regime, and, in particular,showed it converges to the minimum 2-norm solution of the underdetermined problem $AG^{1/2}x = b$, where G is the fixed diagonal preconditioner matrix (Theorem 5.2.1). Moving to full gradient diagonalized Adagrad, we showed convergence in the underdetermined LR regime (Theorem 5.3.6), and proved that when features of our underdetermined matrix A are small in norm, and our step size is small enough, Adagrad will promote the training in the direction of those sparser features, while suppressing the training in the direction of frequent features, when compared to gradient descent (Theorem 5.4.3). Additionally, we showed that one can construct a diagonal matrix D such that weighted least-squares converges to the minimum ℓ_∞-norm solution of underdetermined linear regression (Theorem 5.5.2), and likewise the same result under the correct choice of preconditioner when optimizing with fixed preconditioner gradient descent (Theorem 5.5.3). This allowed us to better understand why Adagrad promotes small LSM scored solutions like the ∞-norm solution does.

The formation of Theorem 5.4.3 came from experimental findings that showed that, under the underdetermined LR regime, and for small enough step sizes, Adagrad promoted diffuse (or non-approximately sparse) solutions when compared with gradient descent. This was measured using a metric derived from compressive sensing which we dubbed the ”local-sparsity metric” that measures approximate sparsity of a vector.

Finally, we show the benefit of such a behaviour in the application of interpolating a line via spiked functions. Namely, we experimentally observe that optimizers that sport a low local-sparsity metric score correlate with smaller residuals when interpolating a line. Specifically, we see that for step size small enough, the solution produced by Adagrad has the best residual results and lowest LSM score when compared to minimum 2-norm, 1-norm, and ∞-norm solutions.

Possible future work on this subject includes the analysis of what sort of solution full gradient Adagrad converges to in the underdetermined LR regime, extensions to the stochastic Adagrad scenario, and as well as studying the algorithmic behaviours and biases of the other adaptive algorithms such as Adam and its stochastic variant in this regime. Extension to studying the algorithmic behaviour to the non-linear setting is another path of possibility as well.

References

[1] Madhu S. Advani and Andrew M. Saxe. High-dimensional dynamics of generalization error in neural networks, 2017.

[2] Kimon Antonakopoulos, Panayotis Mertikopoulos, Georgios Piliouras, and Xiao Wang. AdaGrad avoids saddle points. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 731–771. PMLR, 17–23 Jul 2022.

[3] Larry Armijo. Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics*, 16(1):1 – 3, 1966.

[4] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, jul 2019.

[5] Mikhail Belkin, Daniel Hsu, and Ji Xu. Two models of double descent for weak features. *SIAM Journal on Mathematics of Data Science*, 2(4):1167–1180, jan 2020.

[6] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures, 2012.

[7] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

[8] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004.

[9] David S. Broomhead and David Lowe. Multivariable functional interpolation and adaptive networks. *Complex Syst.*, 2, 1988.

[10] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[11] Martin D. Buhmann. *Radial Basis Functions: Theory and Implementations*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2003.

[12] C. Cartis, N. I. M. Gould, and Ph. L. Toint. On the complexity of steepest descent, newton's and regularized newton's methods for nonconvex unconstrained optimization problems. *SIAM Journal on Optimization*, 20(6):2833–2852, 2010.

[13] N. Cesa-Bianchi, A. Conconi, and C. Gentile. On the generalization ability of on-line learning algorithms. *IEEE Transactions on Information Theory*, 50(9):2050–2057, 2004.

[14] Junyi Chai, Hao Zeng, Anming Li, and Eric W.T. Ngai. Deep learning in computer vision: A critical review of emerging techniques and application scenarios. *Machine Learning with Applications*, 6:100134, 2021.

[15] Rodrigo de Azevedo (https://math.stackexchange.com/users/339790/rodrigo-de azevedo). Does gradient descent converge to a minimum-norm solution in least-squares problems? Mathematics Stack Exchange. URL:https://math.stackexchange.com/q/3499305 (version: 2022-02-18).

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[17] Timothy Dozat. Incorporating Nesterov Momentum into Adam. In *Proceedings of the 4th International Conference on Learning Representations*, pages 1–4.

[18] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 07 2011.

[19] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. ISMP Talk Slides, 2012.

[20] Alexandre Défossez, Léon Bottou, Francis Bach, and Nicolas Usunier. A simple convergence proof of adam and adagrad, 2022.

[21] Kunihiko Fukushima. Cognitron: A self-organizing multilayered neural network. *Biological Cybernetics*, 20(3-4):121–136, 1975.

[22] Andrei Yur'evich Garnaev and Efim Davydovich Gluskin. The widths of a euclidean ball (russian). *Doklady Akademii Nauk SSSR*, 277(5):1048–1052, 1984.

[23] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[24] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[25] Tibshirani Ryan Gordon Geoff. Optimization 10-725: Lecture 5: Gradient descent revisited., 2012.

[26] Robert Mansel Gower, Nicolas Loizou, Xun Qian, Alibek Sailanbayev, Egor Shulgin, and Peter Richtarik. Sgd: General analysis and improved rates, 2019.

[27] Trevor Hastie, Andrea Montanari, Saharon Rosset, and Ryan J. Tibshirani. Surprises in high-dimensional ridgeless least squares interpolation, 2020.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.

[29] Geoffrey Hinton. Neural networks for machine learning: Lecture 6e rmsprop: Divide the gradient by a running average of its recent magnitude. Lecture Notes, 2018.

[30] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[31] Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[32] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[33] Sham Kakade. Cse 546: Machine learning : Lecture 9: Optimization 1: Gradient descent, 2015.

[34] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima, 2016.

[35] Nitish Shirish Keskar and Richard Socher. Improving generalization performance by switching from adam to sgd, 2017.

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

[37] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[38] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, 12 1989.

[39] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. *Efficient BackProp*, pages 9–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[40] Kaifeng Lyu and Jian Li. Gradient descent maximizes the margin of homogeneous neural networks, 2020.

[41] Chui Michael, Hall Bryce, Singla Alex, and Sukharevsky Alex. The state of ai in 2021. https://www.mckinsey.com/capabilities/quantumblack/our-insights/global-survey-the-state-of-ai-in-2021.

[42] Marvin Minsky and Seymour A. Papert. *Perceptrons: An Introduction to Computational Geometry.* The MIT Press, 09 2017.

[43] Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and And Shapiro. Robust stochastic approximation approach to stochastic programming. *Society for Industrial and Applied Mathematics*, 19:1574–1609, 01 2009.

[44] Yurii Nesterov. *Local Methods in Unconstrained Minimization*, page 32–36. Springer, 2004.

[45] OpenAI (2023). Gpt-4 technical report, 2023.

[46] Francesco Orabona. A modern introduction to online learning, 2022.

[47] B.T. Polyak. Gradient methods for the minimisation of functionals. *USSR Computational Mathematics and Mathematical Physics*, 3(4):864–878, 1963.

[48] Herbert Robbins and Sutton Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400 – 407, 1951.

[49] F. Rosenblatt. (1958) F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," [i]Psychological Review[/i], 65:386-408. In *Neurocomputing, Volume 1: Foundations of Research*. The MIT Press, 04 1988.

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

[51] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[52] Vatsal Nilesh Shah. *On Variants of Stochastic Gradient Descent*. PhD thesis, The University of Texas at Austin, 2020.

[53] Daniel Soudry, Elad Hoffer, Mor Shpigel Nacson, Suriya Gunasekar, and Nathan Srebro. The implicit bias of gradient descent on separable data, 2022.

[54] Amirsina Torfi, Rouzbeh A. Shirvani, Yaser Keneshloo, Nader Tavaf, and Edward A. Fox. Natural language processing advancements by deep learning: A survey, 2020.

[55] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[56] Bohan Wang, Qi Meng, Wei Chen, and Tie-Yan Liu. The implicit bias for adaptive optimization algorithms on homogeneous neural networks, 2021.

[57] Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning, 2018.

[58] Lin Xiao and Tong Zhang. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4):2057–2075, 2014.

[59] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network, 2015.

[60] Yu Yaoliang. Cs794/co673: Optimization for data science lec 00: Optimization basics, September 2022.

[61] Yu Yaoliang. Cs794/co673: Optimization for data science lec 01: Introduction, September 2022.

[62] Yu Yaoliang. Cs794/co673: Optimization for data science lec 05: Subgradient algorithms, September 2022.

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

[64] Jingzhao Zhang, Sai Praneeth Karimireddy, Andreas Veit, Seungyeon Kim, Sashank Reddi, Sanjiv Kumar, and Suvrit Sra. Why are adaptive methods good for attention models? In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 15383–15393. Curran Associates, Inc., 2020.

[65] Yin Zhang. A simple proof for recoverability of l1 minimization: Go over or under? 01 2005.

[66] Difan Zou, Yuan Cao, Yuanzhi Li, and Quanquan Gu. Understanding the generalization of adam in learning neural networks with proper regularization, 2021.