The Australian National University
2600 ACT | Canberra | Australia

Australian National University

School of Computing

College of Engineering and Computer Science (CECS)

Applying Stochastic Subgradient Descent to Machine Learning Methods

— 24 pt Honours project (S1 2021–2022)

A thesis submitted for the degree
Bachelor of Advanced Computing

By:
Edmund Hofflin

Supervisor:
Dr. Stephen Gould

June 2022

Declaration:

I declare that this work:

- upholds the principles of academic integrity, as defined in the University Academic Misconduct Rules;
- is original, except where collaboration (for example group work) has been authorised in writing by the course convener in the class summary and/or Wattle site;
- is produced for the purposes of this assessment task and has not been submitted for assessment in any other context, except where authorised in writing by the course convener;
- gives appropriate acknowledgement of the ideas, scholarship and intellectual property of others insofar as these have been used;
- in no part involves copying, cheating, collusion, fabrication, plagiarism or recycling.

June, Edmund Hofflin

Acknowledgements

I am very grateful to Professor Stephen Gould, my supervisor, for giving me the opportunity to study under his direction. During the course of this thesis, he has provided me with significant help and support, without which this thesis would not exist.

I wish to thank Dr. Lindon Roberts for introducing me to the fascinating area of optimisation through his course and then encouraging my continued interest with special lectures and talks. I also wish to thank Edward Smyth and Nicholas Hilderson for engaging with and matching my enthusiasm for computer science, mathematics, and optimisation over the past 5 years.

Finally, I am very appreciative of Dr David Bowen and Dr Tri Phan for their constant medical care since 2019, enabling me to continue enjoying my studies.

Abstract

This thesis collates, summarises, and extends the theory and application of stochastic subgradient descent to the training of large-scale machine learning models. We begin by identifying that standard smooth optimisation techniques are producing models with low robustness and performance, as they assume the models are smooth when most are not. Motivated by this issue, we examine stochastic subgradient descent as a potential alternative. This examination begins by deriving the algorithm as a generalisation of stochastic gradient descent, by substituting the gradient with a generalised derivative, the Clarke subdifferential. Following recent mathematical developments in variational analysis, we then outline the convergence theory of stochastic subgradient descent. In particular, we derive a convergence result that applies to the entire domain of supervised learning, a framework that underpins many state-of-the-art machine learning models. We experimentally verify these results by producing the first generic implementation of stochastic subgradient descent and testing its convergence on a disparate group of functions. Overall, our examination bridges the gap between the theory and practice of stochastic subgradient descent, determining that it is a viable method for training robust non-smooth machine learning models. Finally, we outline the need for theory related to the rate and order of convergence for stochastic subgradient descent and conduct preliminary experimental tests suggesting that the algorithm exhibits Q-sublinear convergence on a wide class of functions.

Table of Contents

Notation and Terminology

We use the following notation and terminology:

Sets

$\mathbb{N} = \{1, 2, \ldots\}$ is the natural numbers. $\mathbb{Z}_+ = \mathbb{N} \cup \{0\}$ extends the natural numbers to include 0. $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ and $\mathbb{C}$ are the integers, rationals, reals and complex numbers, respectively. We also consider the extended reals, $\tilde{\mathbb{R}} = [-\infty, \infty]$.

We use standard set operations, using $\cup$, $\cap$ and $\setminus$. The unary operation A^C is also used to denote the complement, $U \setminus A$, with respect to a universal set U. Additionally, if A is a subset of B with the possibility of equality, then we write $A \subseteq B$. For subsets with no possibility of equality, $A \neq B$, then we use $A \subsetneq B$. Two sets A and B are disjoint if $A \cap B = \varnothing$, where $\varnothing$ denotes the empty set. Finally, $\mathcal{P}(A)$ is the power set of A, the set of all subsets of A. We note that $|\mathcal{P}(A)| = 2^{|A|}$.

$B_r(\mathbf{x}) = \{\mathbf{y} : \|\mathbf{x} - \mathbf{y}\|_2 < r\}$ denotes the open ball of radius $r > 0$ centered at $\mathbf{x} \in \mathbb{R}^d$.

Vectors

We use bold-typeface to denote a vector $\mathbf{x} \in \mathbb{R}^d$. Scalars $k \in \mathbb{R}$ have standard typeface.

The i-th component of a vector is denoted with a superscript, $\mathbf{x}^i$, to distinguish components from algorithm iterates.

We denote the p-norms by $\|\cdot\|_p$. That is:

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{d} \left(\mathbf{x}^i\right)^p \right)^{\frac{1}{p}}$$

We use the notation $\langle \cdot, \cdot \rangle$ to denote the inner product between two vectors.

Functions

For set X and Y, a map $f : X \to Y$, and a subset $A \subseteq X$, we define

$$f(A) = \{y \in Y : \exists a \in A, f(a) = y\}$$

For $B \in Y$, we define

$$f^{-1}(B) = \{x \in X : f(x) \in B\}$$

A function $f : X \to Y$ is injective if and only if $f(x) \neq f(x')$ for all $x \neq x' \in X$. The function is surjective if and only if $f(X) = Y$. Finally, f is bijective if and only if it is both injective and surjective.

A function $f : \mathbb{R} \to \mathbb{R}$ is right-continuous (left-continuous) if $f(x_n) \to f(x)$ as $n \to \infty$ whenever $x_n \downarrow x$ ($x_n \uparrow x$) as $n \to \infty$.

The limit superior and limit inferior of a sequence $\{x_n\}_{n=1}^{\infty} \subseteq \mathbb{R}$ are

$$\limsup_{n\to\infty} = \lim_{n\to\infty} \left(\sup_{k \geq n} x_k \right) \qquad \liminf_{n\to\infty} = \lim_{n\to\infty} \left(\inf_{k \geq n} x_k \right)$$

We note that both the limit superior and limit inferior are always well-defined as (possibly infinite) elements of $\tilde{\mathbb{R}}$, whereas the standard limit does not always exist.

We use $\arg\min_{x \in \mathcal{C}} f(x)$ ($\arg\max_{x \in \mathcal{C}} f(x)$) to denote the argument of the minimum (maximum) of the function f, over the constraint set C.

The notation $f : A \rightrightarrows B$ denotes a set-valued function from A to subsets of B. This is equivalent to $f : A \to \mathcal{P}(B)$.

Chapter 1

Introduction

Machine learning has quickly emerged as a dominant technology in the 21st century. Within the space of two decades, it has progressed from an obscure research area to an established industry that influences government decisions [6]. This growth is expected to not only continue but accelerate, with the machine learning market size predicted to grow from USD 21.17 billion in 2022 to USD 209.91 billion by 2029 [53].

The method that underpins machine learning is learning from experience, employing algorithms and systems that train a model to understand the characteristics and relationships of the chosen problem. This training process is completed by finding the parameters, i.e. settings, of the model that lead to the least number of mistakes and errors in its understanding. This process of finding optimal parameters to minimise the errors can be formalised as a mathematical optimisation problem: if $\mathcal{W}$ is the space of parameters for the model and $\ell : \mathcal{W} \to \mathbb{R}$ is a loss function that numerically summarises the errors produced by a model with set parameters, then the training process becomes:

$$\arg\min_{\mathbf{w} \in \mathcal{W}} \ell(\mathbf{w})$$

Given that training a machine learning model is equivalent to solving an instance of this optimisation problem, techniques and methods from mathematical optimisation can be employed to train and produce machine learning models. The most popular and successful is stochastic gradient descent, a descent method that aims to solve the training optimisation problem by producing a sequence of iterates $\{\mathbf{w_k}\}_{k \geq 0}$ that move 'downhill' towards the optimal choice of parameters. Figure 1.1 illustrates such a method.

In mathematics, 'downhill' is measured by the gradient, its direction pointing in the steepest direction and its magnitude measuring how steep that slope is. As such, stochastic gradient descent, and the vast majority of other descent methods, use the gradient of the loss to choose the next parameter iterate. However, the gradient is only defined if

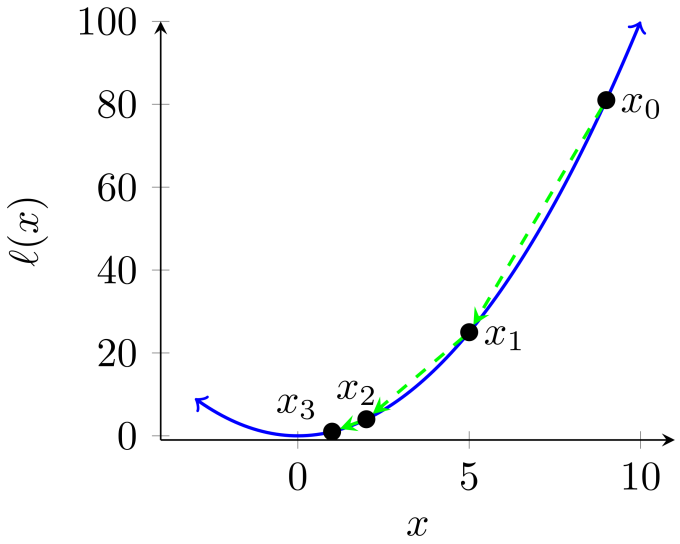


Figure 1.1: Descent Methods Moving 'Downhill' to Find Optimal Parameters

the loss function ℓ is differentiable at the current parameter iterate: if ℓ is not differentiable at the current parameter iterate, then the gradient is undefined and 'downhill' can not be precisely determined. In the case of training a machine learning model, this will limit the performance and robustness of the produced model [29]. Many state-of-the-art machine learning models, such as neural networks, are non-smooth, meaning that they have non-differentiable points and consequently may suffer from poor training.

A possible solution to this issue is stochastic subgradient descent. This algorithm is a descent method that generalises stochastic gradient descent by using a generalised derivative, the Clarke subdifferential, in place of the gradient. The Clarke subdifferential defines a notion of 'downhill' even at non-differentiable points and so overcomes the issues associated with applying smooth descent methods to training non-smooth machine learning models. However, this generalisation comes at a cost: the theory and application of the Clarke subdifferential and stochastic subgradient descent are not well-established, with only a few theoretical results and almost no experimental tests. As a result, the practical application of stochastic subgradient descent to training non-smooth machine learning models is undetermined.

The first step in resolving this issue is to discern the existing theory of stochastic subgradient descent. Fortunately, the two most critical theoretical results have been proven: the derivation of stochastic subgradient descent as a generalisation of stochastic gradient descent, and the conditions of convergence. These results are necessary to understand how the algorithm overcomes the issues associated with smooth descent methods while still effectively solving the optimisation problem. Furthermore, together these results can be used to assess other important aspects of the algorithm which are also necessary for determining its practical viability, such as computability and scope of application.

However, while these two results have been proven, both can only be pieced together from many distinct papers [30, 31, 82, 36, 39]. Therefore, to overcome this problem and make the first step towards determining the practical application of stochastic subgradient descent to the training of non-smooth machine learning models, we undertake a complete and systematic study of this theory.

The next step is to verify the translation of this theory to the real-world, confirming that the mathematical models can be practically implemented and applied. While stochastic subgradient descent has been tested in some limited settings [11, 61], it is yet to be tested in complete generality. Consequently, we implement the first generic version of the algorithm, identifying and resolving the various computational and approximation issues of such an implementation. Using this implementation, we can test the algorithm and its convergence on a wide class of functions, verifying their translation to the real-world.

The final step in determining whether stochastic subgradient descent can be used to train machine learning models is to measure its speed: even if the algorithm produces better models than smooth descent methods, it will not be used if the training time is significantly longer. Stochastic subgradient descent's speed can be measured by proving its rate and order of convergence, however, this has only been done in very limited settings and with no experimental verification [35, 45, 71]. Hence, there is no theory or experimental evidence that can be used to analyse the algorithm's speed in the setting of training machine learning models. We fill this gap by conducting tests providing experimental evidence that stochastic subgradient descent exhibits a rate and order of convergence competitive with stochastic gradient descent on the same wide class of functions on which it converges.

Overall, to determine whether stochastic subgradient descent can be practically applied to the training of non-smooth machine learning models, we undertake three steps in this thesis: firstly, we systematically examine the theory of Stochastic Subgradient Descent; secondly, we navigate and verify the Application of Theory to the real-world; and finally, we conduct experimental tests that suggest the algorithm has a Rate and Order of Convergence, i.e. speed, comparable with the most popular smooth descent method.

1.1 Related Work

Most research on the optimisation of non-smooth machine learning models explores the use of generalised derivatives and the vast majority of that research uses the Clarke Subdifferential. However, there are exceptions. Most notably, the Fénchel subgradient has been used for many years and has found significant use in convex problems: both the Douglas-Rachford splitting method [37] and AdaGrad [38] employ the Fénchel subgradient.

There has also been research into the theory and application of abstract variants of

Stochastic Subgradient Descent that allow the Clarke Subdifferential to be substituted with any generalised derivative. In particular, such an algorithm has proven to converge under comparable conditions to Stochastic Subgradient Descent [85]. Building upon such abstract variants, a recent paper by Jeróme Bolte and Edouard Pauwels has developed a framework based upon conservative fields that provides a rich calculus for generalised derivatives [18]. As a result, this framework is quite powerful, allowing many different types and forms of generalised gradients to be manipulated and modelled. For example, the Clarke Subdifferential can be viewed as a minimal convex conservative field. Consequently, the framework can be used to prove a near-identical convergence result of Stochastic Subgradient Descent to that outlined in this thesis [20]. However, while this framework is quite powerful, it is an entirely theoretical model. As a result, specific generalised derivatives, such as the Clarke Subdifferential, are needed to implement any algorithm, resulting in a reduction to a variant of Stochastic Subgradient Descent. Additionally, the Clarke Subdifferential exhibits specific properties that make it the ideal generalisation for optimisation purposes[1]. Hence, given this thesis' focus on Stochastic Subgradient Descent's application to training machine learning problems, we elected to forgo this framework and examine the non-abstract, Clarke Subdifferential method.

While most research on the optimisation of non-smooth machine learning models explores the use of generalised derivatives, not all does. There are a handful of methods that employ numerical approximation of either the derivative directly, for example, numerical differentiation using backward differences [28], or a surrogate model, for example, automatic differentiation [74]. We do examine these methods briefly, illustrating their shortcomings and consequently why generalised derivatives are the preferred method. In the case of automatic differentiation, we also outline its relationship to Stochastic Subgradient Descent. However, there are also optimisation methods that rely upon neither generalised derivatives nor numerical approximation. These methods fall into the area of 'derivative-free optimisation' and include swarm optimisation [42] and evolutionary algorithms [5]. These methods are very effective when the cost of evaluating the objective function is high. Given that most machine learning models are designed to have computationally cheap inference, these methods have seen more application in the optimisation of the hyperparameters of machine learning models, where the evaluation cost of the objective function is the entire training cycle [80].

Finally, for some historical context, we note that while the Clarke Subdifferential was first defined in 1975 [30], its application to descent methods only began in the early 2000s and became a serious consideration over the past few years. As a result, the nomenclature of the entire area is variable and inconsistent, differing by author and context. For example, the central algorithm Stochastic Subgradient Descent is also known as Clarke stochastic gradient descent, non-smooth stochastic gradient descent, stochastic extragradient descent, and even just stochastic gradient descent [15, 45, 49]. This disparity has led to various pockets of isolated research, leading to some results being derived independently multiple times [89, 45].

[1] See Other Generalised Derivatives for this analysis.

Chapter 2

Background

In this chapter, we outline the background required to understand and follow the central results and experiments of the thesis. We first examine optimisation, outlining the basic concepts and then the theory and methods for smooth problems. Subsequently, we examine machine learning, exploring its applications and formal mathematical problem statements.

2.1 Introduction to Optimisation

This section is dedicated to the fundamental ideas and constructions of optimisation, setting out the formal definitions and terms used throughout the thesis. Our treatment of these concepts aligns closely with Boyd and Vandenberghe's 'Convex Optimisation' [23], although we streamline some results while extending others due to differences in focuses. The section is concluded by formally stating the problems that we consider.

2.1.1 Preliminary Definitions

We begin with the formal definition of an optimisation problem:

Definition 2.1 (Optimisation Problem)**.** *An* Optimisation Problem *is a problem of the following form:*

$$\min_{\mathbf{x}\in\mathcal{C}} f(\mathbf{x}) \tag{2.1.1}$$

where $f : X \subseteq \mathbb{R}^d \to \tilde{\mathbb{R}}$ *is the* Objective Function *from a subset* X *of* $\mathbb{R}^d$ *to the extended reals* $\tilde{R}$*,* $C \subseteq X$ *is the* Constraint Set*, and* $\mathbf{x} \in \mathbb{R}^d$ *is the* Optimisation Variable*.*

We quickly note that the equality

$$\max_{\mathbf{x}\in X} f(\mathbf{x}) = -\min_{\mathbf{x}\in X} -f(\mathbf{x})$$

enables optimisation problems to examine both minimisation and maximisation problems. The explicit maximisation case is eliminated from the definition to simplify the theory and algorithms.

The most critical property of any optimisation problem is its feasibility:

Definition 2.2 (Feasibility)**.** *Suppose we have an optimisation problem of form* (2.1.1)*. A point* $\mathbf{x} \in \mathbb{R}^d$ *is* Infeasible *if* $\mathbf{x} \notin \mathcal{C}$ *or* $f(\mathbf{x}) = \infty$*, and* Feasible *if* $\mathbf{x} \in \mathcal{C}$ *and* $f(\mathbf{x}) < \infty$*. The optimisation problem itself is* Infeasible *if the constraint set* $\mathcal{C}$ *is empty, i.e.* $\mathcal{C} = \varnothing$*, or if* $f(\mathbf{x}) = \infty$ *for all* $\mathbf{x} \in \mathcal{C}$*. Otherwise, the problem is* Feasible.

An obvious question is why one would ever set the constraint set to be empty. While it would be absurd to set $\mathcal{C} = \varnothing$ explicitly, the constraint set is often defined as the union, intersection, and complement of a variety of other sets and so could unintentionally be empty. For example, the following constraint set is not obviously empty:

$$\mathcal{C} = \left\{\mathbf{x} \in \mathbb{N}^3 : \left(\mathbf{x^1}\right)^m + \left(\mathbf{x^2}\right)^m = \left(\mathbf{x^3}\right)^m, m \in \mathbb{N}\backslash\{1,2\}\right\}$$

Infeasibility arises within many real-world applications: for example, the minimal requirements for one resource could be more than is available. In these cases, the problem must be adapted to become feasible, otherwise no further progress can be made.

Once a problem is deemed feasible, we can focus on producing solutions. We will do this by searching for minima:

Definition 2.3 (Global and Local Minima)**.** *Let* $f : X \subseteq \mathbb{R}^d \to \tilde{\mathbb{R}}$*. A vector* $\mathbf{x}^\star \in X$ *is a* Global Minimum *of* f *if* $f(\mathbf{x}^\star) \leq f(\mathbf{x})$ *for all* $\mathbf{x} \in X$*, i.e.* $\mathbf{x}^\star$ *produces the lowest value of* f*.*

Alternatively, a vector $\mathbf{x}' \in X$ *is a* Local Minimum *of* f *if there is some local neighbourhood* $U_{\mathbf{x}'}$ *of* $\mathbf{x}'$ *for which* $f(\mathbf{x}') \leq f(\mathbf{x})$ *for all* $\mathbf{x} \in U_{\mathbf{x}'}$*, i.e.* $\mathbf{x}'$ *is a global minimum of* f *on a restricted region around* $\mathbf{x}'$*.*

Firstly, note that all global minima are local minima. Secondly, note that there can be anywhere between 0 and infinite local and/or global minima: on $\mathbb{R}$, the function $f(x) = e^x$ has no local or global minima, but every point is a local and global minima of $f(x) = 1$. Searching for minima is the standard approach for optimisation methods, as they are good candidates for optimal points.

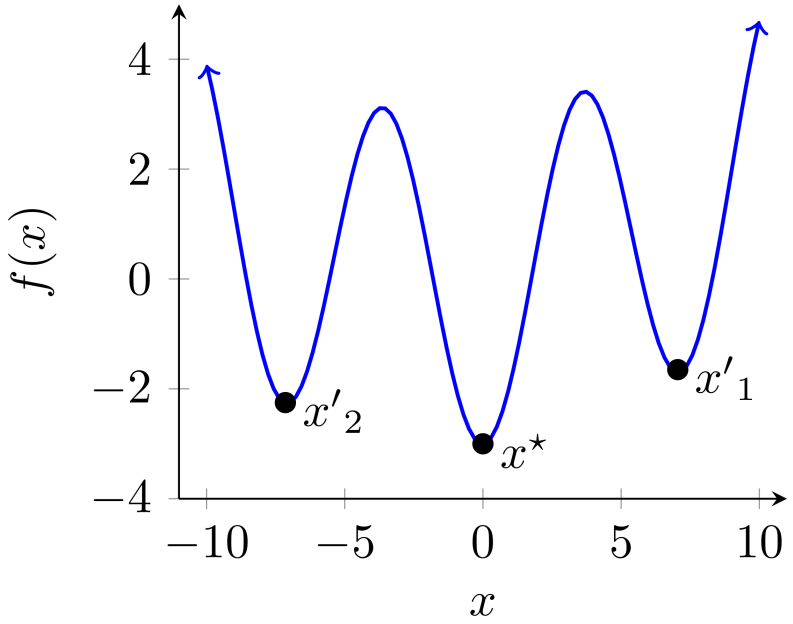


Figure 2.1: Global Minimum $x^\star$ and Local Minima x'_1 and x'_2

Definition 2.4 (Optimal Points)**.** *Suppose we have an optimisation problem of form* (2.1.1)*. Then a vector* $\mathbf{x}^\star \in X$ *is a* Globally Optimal Point *of the optimisation problem if it is a solution, that is:*

$$f(x^\star) = \min_{\mathbf{x}\in\mathcal{C}} f(\mathbf{x})$$

In this case, $p^\star = f(x^\star)$ *is an* Optimal Value*.*

Alternatively, a vector $\mathbf{x}' \in X$ *is a* Locally Optimal Point *of the optimisation problem if it is a solution to the optimisation problem within some local neighbourhood* $U_{\mathbf{x}'}$ *of* $\mathbf{x}'$*, that is:*

$$f(x') = \min_{\mathbf{x}\in\mathcal{C}\cap U_{\mathbf{x}'}} f(\mathbf{x})$$

i.e. $\mathbf{x}'$ *solves the optimisation problem when it is constrained to a region around* $\mathbf{x}'$*. In this case,* $p' = f(\mathbf{x}')$ *is a* Locally Optimal Value*.*

Note that, just as all global minima are local minima, all globally optimal points and values are locally optimal as well. The similarities between minima and optimal points extend to a provable relation:

Proposition 2.1 (Minima are Optimal)**.** *Suppose we have an optimisation problem of the form* (2.1.1)*. If* $\mathbf{x} \in \mathcal{C}$*, i.e. the point is feasible, and is a local/global minima of the objective function* f*, then* $\mathbf{x}$ *is a locally/globally optimal point.*

Clearly globally optimal values are preferred over locally optimal values. However, in many cases finding global minima can be very difficult and computationally expensive. Additionally, various No Free Lunch theorems also prevent any one method from being optimal over all functions, i.e. any method that produces better solutions for one class of functions must necessarily produce poor solutions over a different class of functions [103]. Furthermore, constraints often render global minima infeasible, moving the optimal points elsewhere. As such, locally optimal values are often targeted instead, with the assumption that their respective locally optimal values will be satisfactory approximations of the true global optimal value. While this assumption is generically false, solutions to many real-world applications are well approximated by locally optimal values and so the assumption is not unreasonable.

Note that the above proposition does not imply that locally/globally optimal points are only found at local/global minima. That converse is false. This can be seen by the following counter example:

$$\min_{x \geq 1} x^2$$

Due to the constraint set, this problem has its only globally optimal value at $x = 1$, which is not a local minima of the objective function $f(x) = x^2$. Figure 2.2 illustrates this counter example.

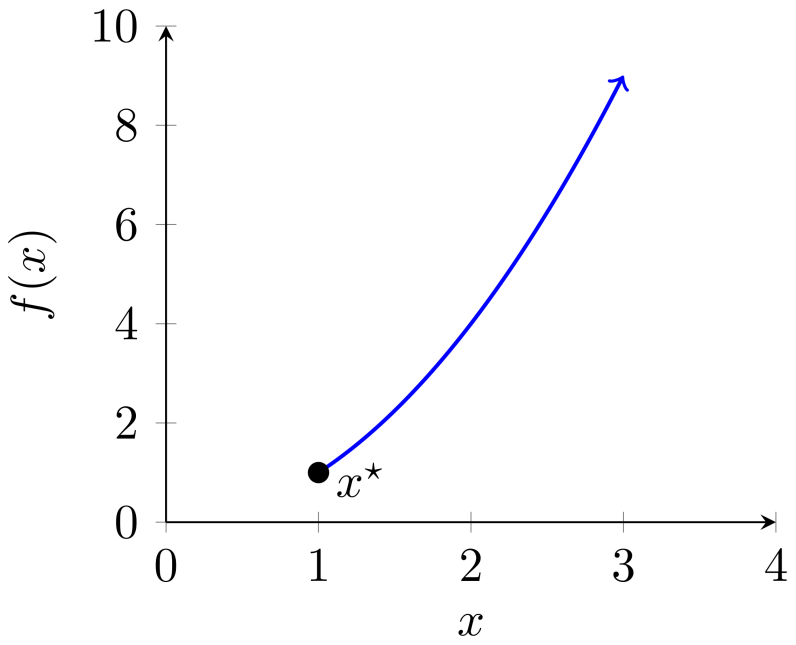


Figure 2.2: Optimal Value of $f(x) = x^2$ with constraint $x \geq 1$

The fundamentals of optimisation are very broad, permitting nearly any minimisation problem. This is deliberate as the definitions must account and allow for all possible types of problems that fall under the umbrella of optimisation, such as combinatorial

optimisation, linear and quadratic programming, and optimal control. However, a field with such wide scope can not be considered in its entirety. So we now introduce some taxonomy to delineate different sub-fields and so enable us to highlight the particular area we consider in this thesis.

2.1.2 Unconstrained and Constrained Optimisation

We begin by considering taxonomy related to the constraint set and domain:

Definition 2.5 (Unconstrained and Constrained Optimisation)**.** *Suppose we have an optimisation problem of form* (2.1.1)*. If* $\mathcal{C} = X$*, i.e. the constraint set is all of* f*'s domain, then the problem is* Unconstrained*. Otherwise, the problem is* Constrained*.*

We note a technicality related to this definition: f's domain need not be the entire space for the problem to be unconstrained. Consequently, when $\mathcal{C} = X \neq \mathbb{R}^d$, the problem is unconstrained even though there are still implicit constraints through the restricted domain. However, if we consider two optimisation problems to be equivalent if the solution to one readily leads to the solution of the other[1], then some implicitly and explicitly constrained problems become equivalent. The following example illustrates this:

$$\min_{x \in \mathbb{R}} \log(x) \qquad \leftrightarrow \qquad \min_{x \geq 0} \log(x)$$

This equivalence suggests that the distinction between constrained and unconstrained optimisation is insignificant. However, that would not be the case. Most often, issues related to constraints, implicit or explicit, can not be eliminated by moving from one form to the other. Instead, these issues are usually repackaged and appear elsewhere. For example, explicit constraints made implicit often force discontinuities into the objective function, which can invalidate many algorithm preconditions. As such, the distinction between unconstrained and constrained optimisation remains significant, although close inspection of the specific objective function and constraint set is often required to properly assess the situation.

The distinction between unconstrained and constrained problems becomes most pertinent when considering optimality conditions. A condition of optimality is feasibility, and so optimal points not only depend upon their function value but their inclusion in the constraint set. This condition is immediately satisfied in unconstrained problems, whereas it must be explicitly considered in constrained problems.

We also quickly note the significance of unconstrained optimisation as a stepping-stone in the development of theory for many constrained optimisation problems. In nearly all cases, the unconstrained relaxation, i.e. setting $\mathcal{C} = X$ but leaving the problem

[1] A complex formal definition can be made for equivalence between optimisation problems [23], but it would be an unnecessary detour for this thesis.

otherwise unchanged, can be theoretically evaluated and solved, whether analytically or algorithmically, with more ease than the original constrained problem. As such, most optimisation theory begins by considering the unconstrained relaxation before assessing the true constrained problem [84].

2.1.3 Continuous and Discrete Optimisation

We outline further taxonomy related to the constraint set and domain:

Definition 2.6 (Continuous and Discrete Optimisation)**.** *Suppose we have an optimisation problem of form* (2.1.1)*. If $\mathcal{C}$ has a connected non-singleton subset, i.e. the optimisation variable can be selected from a continuous interval, on which the objective function f is continuous and finite, then the problem is a* Continuous Optimisation Problem*. Otherwise, the problem is a* Discrete Optimisation Problem*.*

There is no general equivalency between continuous and discrete problems. For example, the following discrete problem

$$\min_{x \in \mathbb{N}} x^2 - 2x + 2$$

can not be transformed into a continuous problem. The problem

$$\min_{x \in \mathbb{R}_+} f(x) := \begin{cases} x^2 - 2x + 2 & x \in \mathbb{N} \\ \infty & \text{otherwise} \end{cases}$$

is equivalent, but it is still a discrete problem as f is not continuous and finite on any connected subset of $\mathbb{N}$. As such, there are significant differences between continuous and discrete optimisation problems. One key difference is that continuous optimisation problems can be solved using methods from calculus and analysis, whereas a discrete problem must rely upon alternative methods, such as cutting-plane or heuristic ant-colony methods [73]. In fact, the techniques from calculus and analysis are so useful that many discrete methods use continuous relaxations, most often done by using the convex hull of the constraint set, to inform their solutions [65].

2.1.4 Smooth and Non-Smooth Optimisation

We now switch focus to the objective function:

Definition 2.7 (Smooth and Non-Smooth Optimisation)**.** *Suppose we have an optimisation problem of form* (2.1.1) *that is a continuous optimisation problem. If there exists some $n \in \mathbb{N}$ such that $f \in C^n$, i.e. f is n-times differentiable for a positive n, then the problem is a* Smooth Optimisation Problem*. If $n = 0$, then the problem is a* Non-Smooth Optimisation Problem*.*

Note that this use of 'smooth' is different to the standard mathematical definition: f need not be infinitely differentiable, only once differentiable, for the optimisation problem to be considered smooth. We also note that non-smooth optimisation is sometimes called 'Derivative-Free Optimisation' [9].

There are no general equivalences between smooth and non-smooth optimisation problems. As such, distinct methods have been developed for solving each problem type. Smooth methods predominately rely upon gradient information to iteratively improve an approximate solution and converge to a minimum. Non-smooth methods are more varied and problem dependent, but reasonably common approaches are: generating smooth models and then approximating the solution using methods from smooth optimisation; developing heuristics to estimate solutions or guide an iterative process; simple (blind) search methods; and lastly, generalised derivative methods, such as the ones we consider in this thesis [9].

Finally but importantly, note that smooth and non-smooth optimisation problems are a refinement of continuous optimisation problems: both smooth and non-smooth problems necessarily have objective functions that are continuous and finite on some subset of $\mathcal{C}$.

2.1.5 Deterministic and Stochastic Optimisation

The taxonomy concerns both the objective function and constraint set:

Definition 2.8 (Deterministic and Stochastic Optimisation)**.** *Suppose we have an optimisation problem of form* (2.1.1)*. If there is any random element in f or $\mathcal{C}$, then the problem is* Stochastic*. Otherwise, the problem is* Deterministic*.*

This definition is very general: the nature of the random element is completely open. Just as with our definition for an Optimisation Problem, this is done to ensure all forms are covered: how the random element is incorporated is very problem and algorithm dependent, and so no general representation can be given. However, despite this vague definition, the distinction between deterministic and stochastic problems is still very clear: if there is any random element, then the problem is stochastic, and otherwise, it is deterministic.

As apparent from the definition, the random element can be embedded in two distinct manners: the objective function and/or the constraint set. Between the two, all aspects of an optimisation problem can be randomised: f can be noisy, the set inclusions $\mathbf{x} \in \mathcal{C}$ and $\mathbf{x} \in X$ could be uncertain, every evaluation of f may be randomly sampled from a family of functions $\{f_i\}_{i \in I}$, and more. That said, by far the most common stochastic element considered in stochastic optimization is a function with noise in its evaluation and derivatives (if they exist). Note that these stochastic elements are not always caused by uncertainty, noise, or error. For example, in large-scale machine learning, stochastic

elements are introduced due computational limitations: the entire dataset can not be processed at once and so batches of disjoint subsets or even individual data points, both of which only model part of the problem, must be considered instead and so contribute stochastic elements [22].

Stochastic elements nearly always make a problem more difficult. As such, motivation is needed to consider such problems. Random elements can arise naturally in an optimisation problem for a variety of reasons: for example, experimental results are never entirely accurate, computational methods must round values, and many real-world requirements can only be estimated. In practice, few real-life processes can be measured or computed deterministically with no error, and so random elements are often unavoidable. This forces us to consider and develop methods for stochastic optimisation.

2.1.6 Convex and Non-Convex Optimisation

The final taxonomy that we are concerned with also relates to both the objective function and constraint set:

Definition 2.9 (Convex Sets)**.** *A set* $S \subseteq \mathbb{R}^d$ *is* Convex *if for all* $0 \leq t \leq 1$ *and* $\mathbf{x_1}, \mathbf{x_2} \in S$ *we have:*

$$t\mathbf{x_1} + (1-t)\,\mathbf{x_2} \in S$$

i.e. S *contains all line segments that join points within* S*.*

Definition 2.10 (Convex Functions)**.** *A function* $f : X \subseteq \mathbb{R}^d \to \mathbb{R}$ *is* Convex *if* X *is convex and for all* $0 \leq t \leq 1$ *and* $\mathbf{x_1}, \mathbf{x_2} \in X$ *we have:*

$$f\left(t\mathbf{x_1} + (1-t)\,\mathbf{x_2}\right) \leq tf\left(\mathbf{x_1}\right) + (1-t)\,f\left(\mathbf{x_2}\right)$$

i.e. the line segment between any two points on the function is never below the function itself.

Definition 2.11 (Convex and Non-Convex Optimisation)**.** *Suppose we have an optimisation problem of form* (2.1.1)*. If* $\mathcal{C}$ *is a convex set and* f *is a convex function, then the problem is* Convex*. Otherwise, the problem is* Non-Convex*.*

In contrast to the previous definitions, this one may seem quite arbitrary and specific. It naturally raises the question: why is convexity significant? The answer to this question is two-fold: firstly, all local minima of a convex function are global minima and so a convex optimisation problem can be solved exactly, rather than approximated, by methods that search for local minima; and secondly, when the constraint set is convex, one can continuously move in a straight line between any two feasible points while remaining

feasible. Together these two facts make convex optimisation a simpler problem than non-convex optimisation.

In many cases[1], additional constraints can be added to produce a convex subproblem. As such, various optimisation techniques target the local subproblem, rather than the overarching global problem. Consequently, the impacts of convexity leads to additional taxonomy distinguishing between the local and global problems:

Definition 2.12 (Local and Global Optimisation)**.** *Suppose we have an optimisation problem of the form* (2.1.1) *and that the convexity of the problem is undetermined. This is then a* Global *optimisation problem. If further constraints are added such that the new constraint set* $\mathcal{C}'$ *is a convex set and* $f|_{\mathcal{C}'}$*, i.e.* f *restricted to* $\mathcal{C}'$*, is now a convex function, then this is a* Local *optimisation subproblem,with respect to the global problem.*

Note that the use of 'local' and 'global' in this definition is distinct from their use in minima and optimal points/values. In this definition, the terms refer to the scope of the entire problem, whereas previously they referred to the properties of a particular point. As such, global optimisation techniques apply to global optimisation problems, rather than being methods that target globally optimal points/values. The former is usually denoted by explicitly stating that the given technique targets global solutions[2]. In fact, global optimisation methods are actually more likely to target local optimal points rather than global optimal points, as there is no convexity assumption to ensure the existence of a unique optimal value.

2.1.7 Our Focus and Assumptions

For the majority of the thesis, we consider the following subset of optimisation problems: global, unconstrained (explicitly and implicitly), continuous, non-smooth, and stochastic problems. That is, we will consider problems of the form:

$$\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x}) \tag{2.1.2}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ is a continuous function that can not be evaluated deterministically (the exact nature of the random element will depend on the specific situation we are considering). We make no other assumptions regarding f or the nature of its optimal points.

There are only a few exceptions when we will break from this form and consider alternative optimisation domains. The most notable exception is the next section, in which

[1] Specifically, when f is a continuously twice-differentiable function. See Second-Order Optimality Condition for Smooth Optimisation.

[2] This nomenclature tends to hold throughout the optimisation community, but there are exceptions.

we outline the theory for global, unconstrained, continuous, *smooth*, stochastic optimisation. All other exceptions are merely comments or explanations of background and associated material. All these exceptions are carefully sign-posted and their discrepancy is motivated to ensure that no false assumptions are made. Otherwise, we will always assume that the optimisation problems are of the form above.

2.2 Smooth Optimisation

The theory and algorithms for non-smooth optimisation rely significantly upon their smooth counterparts, at least in intention and concept. Consequently, we now dedicate time to outline the direction and fundamentals of smooth optimisation. Note that aside from alternating our smoothness assumption, all other assumptions remain for our optimisation problems. That is, we will consider global, unconstrained, continuous, *smooth*, and stochastic optimisation problems of the form

$$\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x}) \tag{2.2.1}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ is a once differentiable function whose values can only be estimated:

$$\overline{f}(\mathbf{x}) \leftarrow f(\mathbf{x}) + \chi$$
$$\overline{\nabla f}(\mathbf{x}) \leftarrow \nabla f(\mathbf{x}) + \xi$$

where $\leftarrow$ denotes calculation/computation/measurement and χ and ξ are random variables.

2.2.1 Smooth Optimality Conditions

As noted after defining Smooth and Non-Smooth Optimisation, smooth optimisation methods predominately rely upon gradient information. Principally, this is due to the optimality conditions related to stationary points:

Definition 2.13 (Stationary Point)**.** *Let $f : X \subseteq \mathbb{R}^d \to \mathbb{R}$ be a once differentiable function. Then $\mathbf{x} \in X$ is a* Stationary Point *of f if $\nabla f(\mathbf{x}) = 0$.*

As the Figure 2.3 suggests, there is a relationship between stationary points and local minima. Fermat made this relationship precise when developing infinitesimal calculus [41]:

Theorem 2.1 (Fermat's Stationary Point Theorem)**.** *Let $f : X \subseteq \mathbb{R}^d \to \mathbb{R}$ be differentiable at $\mathbf{x} \in X$. If $\mathbf{x}$ is a minimum or maximum of f, then $\nabla f(\mathbf{x}) = 0$, i.e. $\mathbf{x}$ is a stationary point of f.*

The contrapositive of the theorem leads to the following optimality condition:

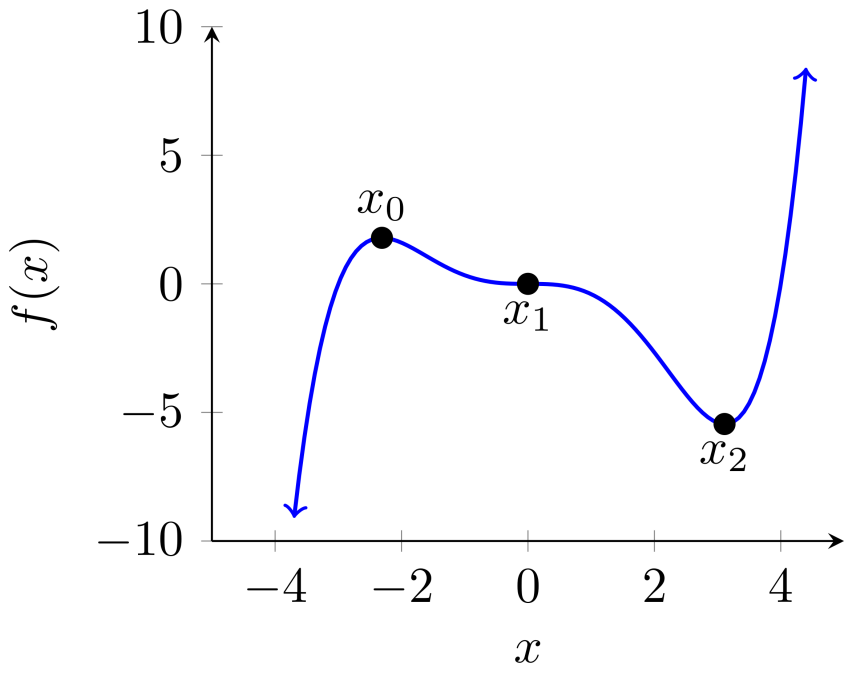


Figure 2.3: Stationary Points x_0, x_1, and x_2. x_0 is a local maximum and x_2 is a local minimum. x_1 is an inflection point and so is neither a maximum or minimum.

Proposition 2.2 (First-Order Optimality Condition for Smooth Optimisation)**.** *([23], Chp. 4, Sec. 2, Subsec. 3) Suppose we have an optimisation problem of the form (2.2.1) and let* $\mathbf{x} \in \mathbb{R}^d$*. If* $\mathbf{x}$ *is not a stationary point, i.e* $\nabla f(\mathbf{x}) \neq 0$*, then* $\mathbf{x}$ *is not an optimal point.*

We strongly emphasise that this optimality condition only applies to our particular flavour of unconstrained, continuous, smooth optimisation problems: the proof in Stephen Boyd and Lieven Vandenberghe's 'Convex Optimisation' outlines how it will fail if the conditions are not satisfied [23]. Such a strong result does not apply to more general settings.

The First-Order Optimality Condition for Smooth Optimisation allows us to discard any non-stationary point for consideration as an optimal point. So the set of potential solutions reduces from all feasible points to a much smaller set. However, not all stationary points are minima: stationary points can also be maxima or neither, as Figure 2.3 illustrates. We could determine whether a stationary point is a minimum or maximum by sampling its local neighbourhood, but practically this method relies upon specific function knowledge. Figure 2.4 illustrates some of those issues.

Fortunately, we can rigorously eliminate maxima by checking the second derivative:

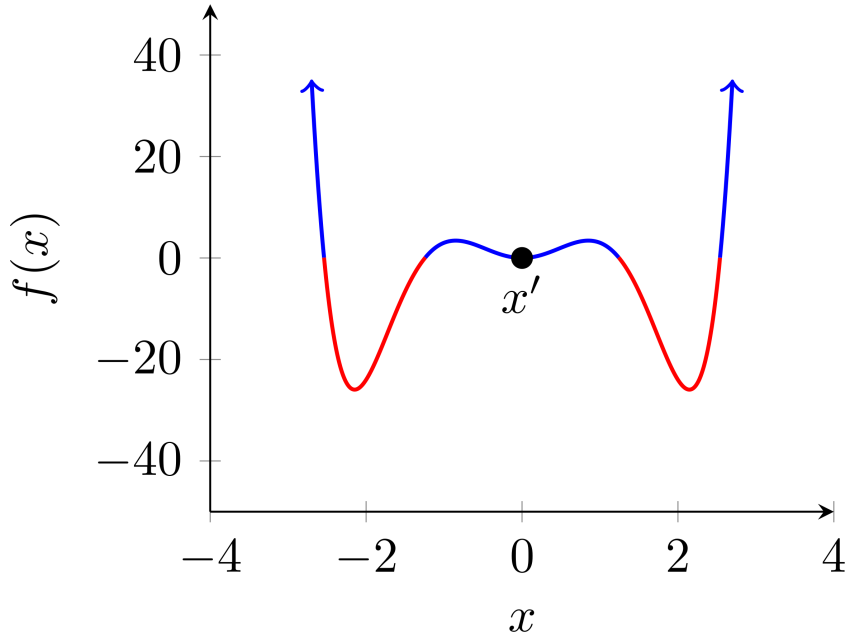


Figure 2.4: $x' = 0$ is a stationary point of $f(x) = x^6 - 8x^4 + 10x^2$. If we use sampling to determine whether x' is a local minimum or maximum, then we wil get different, or even conflicting, results depending on whether we sample in the blue or red regions.

Proposition 2.3 (Second-Order Optimality Condition for Smooth Optimisation)**.** *Suppose we have an optimisation problem of the form (2.2.1) and that f has continuous second partial derivatives. Let $\mathbf{x} \in \mathbb{R}^d$. If $\mathbf{x}$ is a stationary point of f and $\nabla^2 f(\mathbf{x})$ is positive semi-definite[a], then $\mathbf{x}$ is a locally optimal point of f.*

[a] A symmetric matrix M is positive semi-definite if all of its eigenvalues are non-negative.

The second derivative being positive semi-definite forces f to be convex on a local neighbourhood of $\mathbf{x}$, and so if $\mathbf{x}$ is also a stationary point then we necessarily have a minimum. This is a strong result, but it does require a significant additional assumption for the objective function, namely f is continuously twice differentiable at the stationary point of interest. However, the strength of this result does allow us to explicitly state the set of optimal points for such problems:

$$\left\{\mathbf{x} \in \mathbb{R}^d : \mathbf{x} \text{ is a stationary point of } f,\ \nabla^2 f(x) \text{ is positive semi-definite}\right\} \tag{2.2.2}$$

Unfortunately, there are no further optimality conditions that maintain generality in smooth optimisation. In particular, distinguishing between locally and globally optimal points requires additional conditions that are too restrictive, e.g. convexity or pseudo-convexity, and so we must make do with locally optimal points as approximate solutions.

2.2.2 Smooth Optimisation Algorithms and Stochastic Gradient Descent

The First-Order Optimality Condition for Smooth Optimisation and Second-Order Optimality Condition for Smooth Optimisation outline the set of locally optimal points. One might naïvely assume that these conditions are sufficient for producing solutions to smooth optimisation problems as we can implement the following algorithm:

Algorithm 1: Blind Search

input : Objective function $f : \mathbb{R}^d \to \mathbb{R}$ that is continuously twice differentiable.
output: Locally optimal point $\mathbf{x}^\star \in \mathbb{R}^d$ and value $p^\star$.

Initiate set of tested points;
$T \leftarrow \varnothing$;
Loop until an optimal point is found;
while *True* **do**
    Randomly sample point;
    Sample $\mathbf{x} \in \mathbb{R}^d \backslash T$;
    Check optimality conditions;
    if $\nabla f(\mathbf{x}) = 0$ *and* $\nabla^2 f(\mathbf{x})$ *is positive semi-definite* **then**
        Return locally optimal point and value;
        $\mathbf{x}^\star \leftarrow \mathbf{x}, p^\star \leftarrow f(\mathbf{x})$;
        return $\mathbf{x}^\star, p^\star$;
    else
        Update tested points;
        $T \leftarrow T \cup \{\mathbf{x}\}$;

However, an examination of this algorithm shows a raft of issues. Most significantly, the algorithm does not systematically search for a solution, but instead blindly guesses potential solutions. As a result, there are no guarantees that this algorithm will ever find an optimal value. For example, if $f(x) = x^2$ then $x^\star = 0$ is the unique globally optimal value. But the probability that this value will be randomly sampled from the number line is zero[3], and so Blind Search will never actually solve this problem.

The guarantee of a solution, or at least an approximation, is paramount for real-world applications: wasted resources are costly and the absence of a solution could disrupt the system that depends upon it. As such, we now restrict our consideration to methods with guarantees. Specifically, we examine methods that will produce, at worst, an approximation of a local minimum under reasonable conditions. Such methods for smooth optimisation are predominately 'iterative methods'[4], producing a sequence of points,

[3] A formal proof of this probability requires measure theory [44].
[4] For constrained smooth optimisation, many methods are not iterative: the simplex method [34] is a famous example of a finite step method that applies to constrained linear optimisation problems. However for unconstrained smooth optimisation, the vast majority of effective methods are iterative.

called iterates, that converge to an optimal solution $\{\mathbf{x_0}, \mathbf{x_1}, \mathbf{x_2}, \ldots\}$. The most famous of these methods is Newton's method [68] and the collection of descent methods [83].

Newton's method is a second-order method that iteratively solves linear approximations of the stationary point equation $\nabla f(\mathbf{x_k}) = 0$:

Algorithm 2: Newton's Method

input : Optimisation problem of form (2.2.1) with the additional condition that f is twice differentiable and can only be estimated with noise ζ.
Initial point $\mathbf{x_0} \in \mathbb{R}^d$. Step limit $N \in \mathbb{N}$. Step size sequence $\{\alpha_k \in \mathbb{R}\}_{k=0}^{N}$.

output: Approximate locally optimal point $\mathbf{x}^\star \in \mathbb{R}^d$ and approximate locally optimal value $p^\star \in \mathbb{R}$.

Repeat iterative updates;
for $k = 1, 2, \ldots, N$ **do**
  Compute approximate gradient;
  $\mathbf{g_k} \leftarrow \nabla f(\mathbf{x_k}) + \xi_k$;
  Compute approximate Hessian;
  $h_k \leftarrow \nabla^2 f(\mathbf{x_k}) + \zeta_k$;
  Set new iterate;
  $\mathbf{x_{k+1}} \leftarrow \mathbf{x_k} - \alpha_k {h_k}^{-1} \mathbf{g_k}$;
Return locally optimal point and value;
$\mathbf{x}^\star \leftarrow \mathbf{x_N}, p^\star \leftarrow f(\mathbf{x_N})$;
return $\mathbf{x}^\star, p^\star$;

Newton's method is a good algorithm, converging to an optimal point under very reasonable assumptions[5] Furthermore, Newton's method converges quickly[6] under slightly stronger conditions [22]. However, Newton's method does have some shortcomings: most significantly, the additional requirement of a second derivative is often problematic, either because one does not exist or because it is too computationally expensive to calculate or invert. There are many variants of Newton's method that endeavour to overcome these issues: most notably due to its popularity, Adam [57] approximates high-order curvature information using the gradient to reduce the computational and smoothness requirements. As a result, Newton's method and its variants are commonly applied to smooth optimisation problems.

Descent methods are a group of optimisation methods that all aim to improve the current solution by moving downhill in each successive iteration: Figure 2.5 illustrates this concept.

This iterative goal is the only requirement of a descent method and so many different variants exist. For global, unconstrained, continuous, smooth, stochastic optimisation,

[5] Specifically: f has a Lipschitz Hessian and the random variables χ, ξ, and ζ have zero mean and bounded second moments, i.e $\mathbb{E}\left[\|\chi\|^2\right] \leq M_\chi$ for some $M_\chi \in \mathbb{R}$ and similarly for ξ and ζ.

[6] Specifically, order 2 Q-convergence. See Q Convergence for the technical definition.

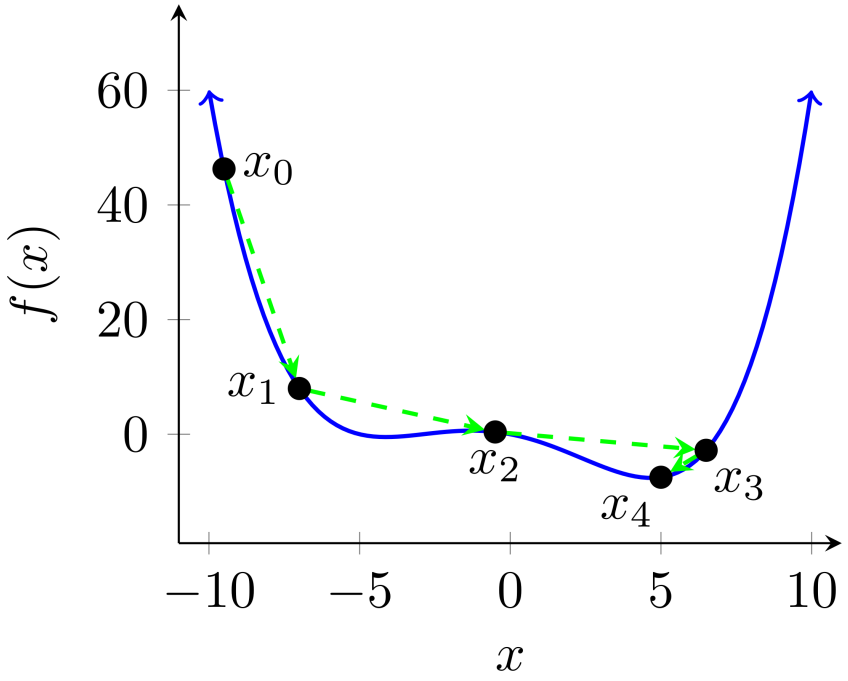


Figure 2.5: Stepping 'downhill' towards a local minimum

easily the most famous and successful method is stochastic gradient descent:

Algorithm 3: Stochastic Gradient Descent

input : Optimisation problem of form (2.2.1). Initial point $\mathbf{x_0} \in \mathbb{R}^d$. Step limit $N \in \mathbb{N}$. Sequence of step sizes $\{\alpha_k \in \mathbb{R}\}_{k=0}^{N}$.

output: Approximate locally optimal point $\mathbf{x}^\star \in \mathbb{R}^d$ and approximate locally optimal value $p^\star \in \mathbb{R}$.

Repeat iterative updates;
for $k = 1, 2, \ldots, N$ **do**
  Compute approximate gradient;
  $\mathbf{g_k} \leftarrow \nabla f(\mathbf{x_k}) + \xi_k$;
  Set new iterate;
  $\mathbf{x_{k+1}} \leftarrow \mathbf{x_k} - \alpha_k \mathbf{g_k}$;
Return locally optimal point and value;
$\mathbf{x}^\star \leftarrow \mathbf{x_N}, p^\star \leftarrow f(\mathbf{x_N})$;
return $\mathbf{x}^\star, p^\star$;

In many ways, stochastic gradient descent is the most simple of descent methods: once an initial point $\mathbf{x_0}$ and step sizes $\{\alpha\}_{k=0}^{N}$ are chosen, the algorithm will follow the gradient downhill and run to completion. Despite this simplicity, the algorithm is very robust and converges in very general and reasonable settings.

Specifically, we have the following three assumptions:

Assumption 2.1 (Lipschitz-Continuity of Objective Gradient)**.** *([22], Chp. 4, Sec. 1, Asm. 1) Given an optimisation problem of the form (2.2.1), we assume that the objective function $f : \mathbb{R}^d \to \mathbb{R}$ is continuous differentiable and its gradient $\nabla f : \mathbb{R}^d \to \mathbb{R}^d$ is Lipschitz continuous with Lipschitz constant $L > 0$, i.e.*

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\|_2 \leq L \|\mathbf{x} - \mathbf{y}\|_2, \qquad \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^d$$

This first assumption ensures the objective function has a continuous derivative and that derivative can only get so steep, i.e. it does not reach infinity at any point and so make the iterative loop undefined.

Assumption 2.2 (First and Second Moment Limits of Errors)**.** *([22], Chp. 4, Sec. 1, Asm. 3) Given an optimisation problem of the form (2.2.1) and iterates $\{\mathbf{x_k}\}$ produced by applying Stochastic Gradient Descent, we assume that:*

1. *The sequence of iterates $\{\mathbf{x_k}\}$ is contained in an open set over which the objective function is bounded below by a scalar f_{inf}.*
2. *There exists scalars $\mu_G \geq \mu > 0$ such that, for all $k \in \mathbb{N}$:*

$$\nabla f(\mathbf{x_k})^T \mathbb{E}_{\xi_k}[\mathbf{g_k}] \geq \mu \|\nabla f(\mathbf{x_k})\|_2^2$$
$$\|\mathbb{E}_{\xi_k}[\mathbf{g_k}]\|_2 \leq \mu_G \|\nabla f(\mathbf{x_k})\|_2$$

3. *There exists scalars $M \geq 0$ and $M_V \geq 0$ such that, for all $k \in \mathbb{N}$,*

$$\mathbb{V}_{\xi_k}[\mathbf{g_k}] \leq M + M_V \|\nabla f(\mathbf{x_k})\|_2^2$$

This second assumption forces three conditions: firstly, the objective function must have a minimum, rather than just tending to $-\infty$; secondly, the gradient approximations $\mathbf{g_k}$ roughly agree with the actual gradient values $\nabla f(\mathbf{x_k})$ in relation to their norms; and finally, that the variance, i.e. noise from the mean, is very gently limited and so the iterates do not just bounce around randomly forever.

Assumption 2.3 (Step Size Sequence Convergence)**.** *Given the Stochastic Gradient Descent, we assume that the sequence of step sizes $\{\alpha_k\}$ satisfy:*

$$\sum_{k=0}^{\infty} \alpha_k = \infty \qquad \sum_{k=0}^{\infty} {\alpha_k}^2 < \infty$$

This final assumption ensures that the step sizes allow the algorithm to explore the entire domain $\mathbb{R}^d$, enabling them to reach a local minimum no matter how away from the initial point it is, but also ensures that the algorithm will still converge.

If these three assumptions are satisfied, then Stochastic Gradient Descent will converge:

Theorem 2.2 (Convergence of Stochastic Gradient Descent)**.** *([22], Chp. 4, Sec. 3, Thm. 9) Suppose we have an optimisation problem of form (2.2.1) and that the Assumptions Lipschitz-Continuity of Objective Gradient, First and Second Moment Limits of Errors and Step Size Sequence Convergence all hold. Then:*

$$\lim_{k\to\infty} \inf \mathbb{E}\left[\left\|\nabla f\left(\mathbf{x_k}\right)\right\|_2^2\right] = 0$$

i.e. the iterates $\{\mathbf{x_k}\}$ will converge to a stationary point, which is a minimum and so a locally optimal point.

This robustness and simplicity are the primary advantages of stochastic gradient descent: it can be implemented and applied in most circumstances with relative ease, yet it will still produce optimal points.

However, stochastic gradient descent does have a few issues. Firstly, while convergence is guaranteed, it will be to any optimal point. As such, the algorithm often produces locally optimal points rather than globally optimal points. As previously discussed, finding globally optimal points is sometimes intractable [103] and so most locally optimal points are satisfactory. However, an issue arises when the local minima are very shallow: Figure 2.6 illustrates such a situation.

In these cases, it would be better for the algorithm to move over the shallow minima and reach a better solution. Various variants exist that aim to overcome this issue: two notable examples are descent methods using momentum [78] and accelerated descent methods [72], both of which incorporate previous gradient and iterative information to select better points. However, these methods do increase computation time and at times produce worse results. Overall, we see that the results of No Free Lunch Theorems are unavoidable and will plague any algorithm, including stochastic gradient descent [103].

Secondly, and more specifically related to stochastic gradient descent, there can sometimes be issues determining appropriate step sizes. As stated in Convergence of Stochastic Gradient Descent, the step sizes must satisfy certain assumptions, but the specific choice does impact the rate and order of convergence, i.e. how quickly the algorithm will reach an optimal point. Some variants overcome this issue by incorporating linesearches, an internal algorithm that greedily optimises the choice of step size at each iteration [90]. However, linesearches do add significant computational requirements to each iteration and they can exacerbate other issues, such as settling in poor locally optimal points.

Despite these and other issues, stochastic gradient descent remains an incredibly popular

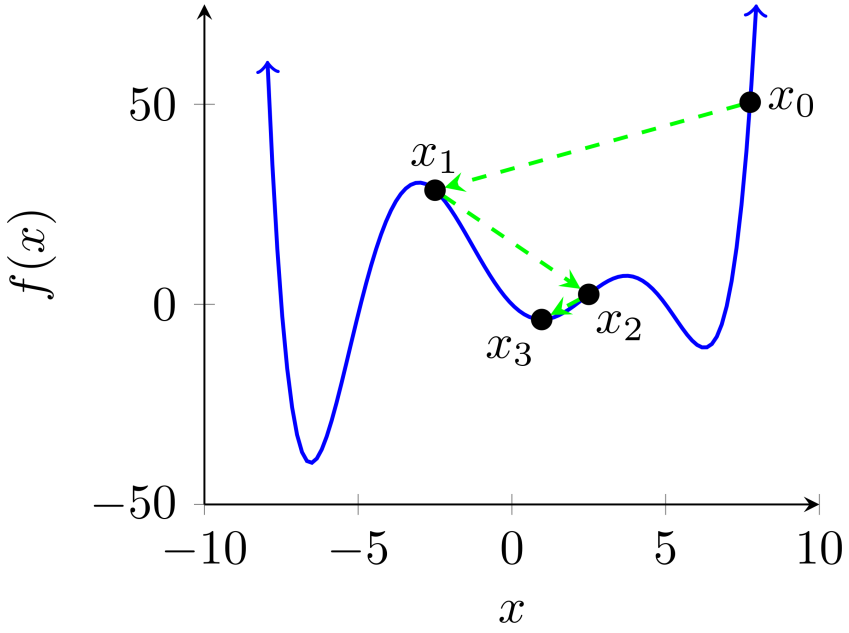


Figure 2.6: Algorithm converges to a local minimum, but passes over a better locally optimal solution and never nears the globally optimal solution.

algorithm for solving global, unconstrained, continuous, smooth, stochastic optimisation problems.

2.3 Machine Learning

In this section, we formally introduce machine learning. After providing a brief summary of the field, we present the mathematical representations of many popular supervised machine learning models, demonstrating that they are optimisation problems. Finally, we classify these optimisation problems, highlighting that they are of the form that we are considering, i.e. global, unconstrained, continuous, non-smooth, stochastic optimisation problems.

2.3.1 Summary of Machine Learning

Machine learning is the study and development of computational models that progressively learn through experience. Encompassing both the theory and application, machine learning primarily focuses on two questions: What rules, laws, and mechanics govern models that learn? And, how can such models be constructed? Machine learning does not just concern itself with artificial models designed and developed by computer scientists but instead examines and considers all possible systems that learn and develop, such as evolutionary systems, the human mind, and larger social networks that adapt

and respond to environmental changes.

Given the depth and complexity of the primary questions and the breadth of systems considered, machine learning must employ methods and techniques from a wide array of sources. As such, machine learning itself is an interdisciplinary field that relies upon computer science, mathematics, statistics, informatics, biology, cognitive science, philosophy, and more.

Just as the theory touches many different areas, machine learning has seen application across a litany of domains. Some examples are: clustering and outlier detection methods being used to aid particle identification and event selection within experimental physics [27]; medicine employing image identification systems to analyse scans and diagnose patients [108]; reinforcement agents autopiloting cars and other transport systems [12]; novel music composed by generative machine learning models [93]; and, techniques inspired by natural evolutionary processes being used to refine and optimise existing machine learning frameworks and improve their performance [110]. Furthermore, machine learning sits at the centre of modern data analysis and artificial intelligence, defining the current practices and paving the path for new methods and techniques [54, 50]. These applications support and drive significant continuous growth in the machine learning industry: it has the potential to deliver an additional USD 30 trillion to the global economy by 2030 [24].

Two factors have led to this phenomenal surge in machine learning research, development, and application. Firstly, the modern ubiquity of data and the emergence of 'big data' has provided the requisite information and statistics needed to train, evaluate, and analyse artificial machine learning models. Secondly, the continuation of Moore's Law [88] has driven down the cost of computation, enabling larger and more complex models to be practically developed and examined. Before 'big data' and cheap computation, machine learning was largely limited to theoretical considerations, with all experimental data being sourced from naturally learning systems and very limited ability to model those systems. Now anyone can spin up new machine learning models using widely available systems and frameworks [75, 76, 7] and access data with great ease [60], streamlining the research and development process and significantly reducing its cost.

While these two factors have certainly galvanised the wider machine learning discipline, they have been particularly impactful on artificial learning systems. Models such as logistic regression, Bayes classifiers, support vector machines, and neural networks have seen significant development and application within the last decade, and new models and concepts are constantly being examined and reviewed [54]. Similarly, such models are also the primary application of machine learning, forming the basis of most prediction, classification, analysis, and management tools [79]. We focus on this sub-area, restricting our attention to the development and application of such artificial learning systems.

2.3.2 Mathematical Representations of Machine Learning Models

We now examine a few of the most popular machine learning models and demonstrate how each can be represented as an optimisation problem.

We limit our attention to machine learning models of the supervised type. There are many other types of machine learning - unsupervised, semi-supervised, reinforcement, dimensionality reduction, to name a few [86] - but we focus on supervised machine learning for three reasons: firstly, it is the most common form of machine learning and so representative of the larger discipline [54]; secondly, many other types of machine learning are extensions of supervised learning, e.g. unsupervised neural networks are usually particular network architectures that use the same data for input and output [51]; finally, our larger focus is on optimising of such problems, rather than on cataloguing them, and so we restrict our attention for the sake of brevity.

Despite this more narrow focus, the models examined are still representative of the entire class of machine learning models: except for a few outliers, machine learning models are frameworks for optimisation problems. Given this, machine learning is completed by applying optimisation techniques. This is one of the primary applications of optimisation [22] and so serves as our primary research motivation.

There are similar treatments of machine learning models and their optimisation representations: 'Optimization Algorithms for Data Analysis' by Wright is such an example [66]. Our treatment differs from others in its focus: we emphasise, rather than avoid, non-smooth components of each form and model, even going so far as to examine a handful of state-of-the-art non-smooth extensions.

General Supervised Learning

Supervised machine learning aims to learn a function that accurately models a relation between inputs and outputs. This function can then be used to predict future outputs or gain insight into the relation.

A supervised machine learning problem has the following form: We have some relation defined by a joint probability distribution $P(\mathbf{x}, \mathbf{y})$ over the input space $\mathcal{I}$ and the output space $\mathcal{O}$. A distribution, rather than an explicit function, is used to allow for uncertainty or stochasticity within the relation. We also have a loss function $\ell : \mathcal{O} \times \mathcal{O} \to \mathbb{R}$ that measures the difference between a prediction $\hat{\mathbf{y}}$ and true outcome $\mathbf{y}$. If we have a function $f : \mathcal{I} \to \mathcal{O}$, then we can calculate its expected loss, also called risk, over the relation:

$$\mathbb{E}_P \left[l \left(f \left(\mathbf{x} \right), \mathbf{y} \right) \right] = \int \ell \left(f \left(\mathbf{x} \right), \mathbf{y} \right) dP \left(\mathbf{x}, \mathbf{y} \right)$$

The general problem of supervised machine learning is to find the function f that minimises this expected loss:

$$\arg\min_{f} \mathbb{E}_P \left[\ell \left(f \left(\mathbf{x} \right), \mathbf{y} \right) \right]$$

This problem is entirely intractable: the space of all functions is too large and lacks structure to systematically minimise over, the result depends upon the unknown relation, and often the calculation is incomputable. However, we can overcome these issues by using the following 'empirical risk' approximation:

$$\arg\min_{\mathbf{w}\in\mathcal{W}} \frac{1}{k}\sum_{i=1}^{k} \ell\left(f\left(\mathbf{x_i};\mathbf{w}\right),\mathbf{y_i}\right)$$

where $\ell : \mathcal{O}\times\mathcal{O}\to\mathbb{R}$ is a loss function, $\mathcal{W}\subseteq\mathbb{R}^d$ is the space of parameters that parametrises the class of functions $f : \mathcal{I}\times\mathcal{W}\to\mathcal{O}$, and $\mathcal{D} = \{(\mathbf{x_i},\mathbf{y_i})\}_{i=1}^{k}$ is a set of input-output pairs that are identically and independently drawn, usually through experimentation, from the joint distribution $P(\mathbf{x},\mathbf{y})$. So we have approximated the true expected loss in two manners: firstly, we are using k data pairs to model the distribution, eliminating the dependency upon the true relation; and secondly, rather than minimising over the entire space of functions, we have instead selected a parametrised class of functions $\{f(\cdot;\mathbf{w})\}_{\mathbf{w}\in\mathcal{W}}$ and are only minimising over this class. If we add the requirement that the class of functions $\{f(\cdot;\mathbf{w})\}_{\mathbf{w}\in\mathcal{W}}$ is computable, then we have resolved all issues with the true expected loss.

Minimising the empirical risk over a particular class of functions is the underlying method for the vast majority of supervised machine learning models. Overall, we get the following pipeline:

1. Data $\mathcal{D}$ is independently and identically drawn from $P(\mathbf{x},\mathbf{y})$.
2. Exploratory data analysis is done, e.g. plotting the data or calculating its basic statistical properties, to inform which class of function $\{f(\cdot,\mathbf{w})\}_{\mathbf{w}\in\mathcal{W}}$ and its parameters $\mathcal{W}$ are most appropriate for the function, e.g. linear functions are appropriate for linear data, not high-order data.
3. A loss function ℓ that suits the data is selected, e.g. 2-norms are common for continuous data but are not suited for categorical data.
4. An optimisation method is used to find an optimal value of empirical risk and its optimal point $\mathbf{w}^\star$ can be used to produce the final function $f^\star(\cdot) = f(\cdot,\mathbf{w}^\star)$.

For the most part, this pipeline is reasonably effective and simple to complete: we can teach systems to learn unknown relations.

However, there are still some issues. There are a handful of smaller complications - drawing data without bias can be difficult, especially if done through experiments [67], and selecting the loss function can sometimes prove difficult for complicated problems like natural language processing [46] - but there is also one significant issue. By minimising the empirical risk, we may overfit the system to the data, reducing its accuracy on the entire distribution: Figure 2.7 illustrates an example of overfitting.

There are various methods to solve this issue: early-stopping - simply halting the optimisation algorithm at an earlier step - is remarkably effective in some situations, stopping

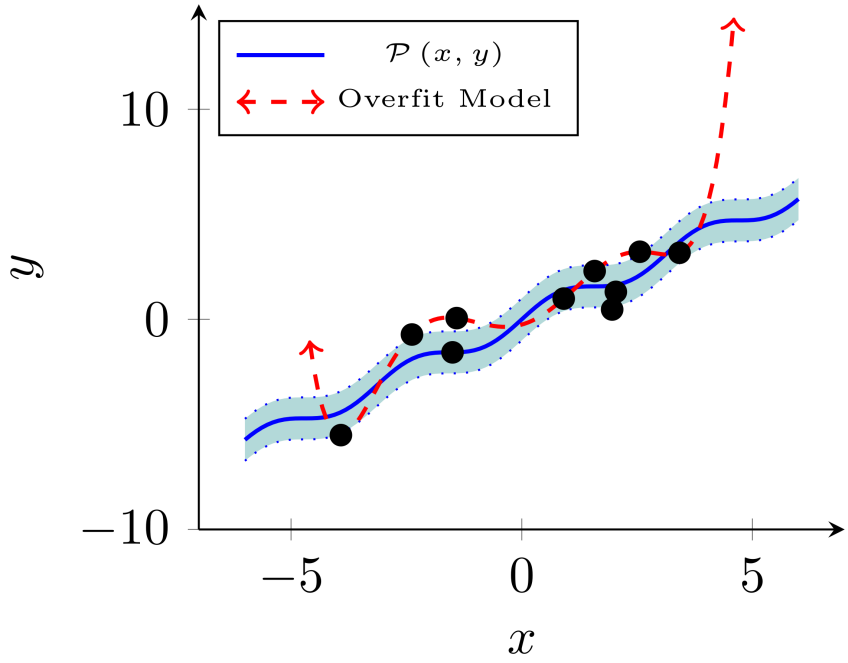


Figure 2.7: Overfitting to the data: The distribution $\mathcal{P}(x, y)$ is given by $y = \frac{\sin(2x)}{2} + x + \mathcal{N}(0, 1)$ where $\mathcal{N}(\mu, \sigma^2)$ denotes the normal distribution with mean μ and variance σ^2. Given a sample of $k = 10$ points from $x \in [-4, 4]$, we can train a model to minimise the empiricial risk. However, this model has overfit the system and so has poor performance outside of the dataset.

the system from settling into a minimum that only models the data rather than the relation [77]; network-reduction aims to lessen the impact of noise within small datasets by pruning unnecessary data, preventing the system from learning the random noise rather than the underlying relation [43]; and, finally the training set can be artificially expended by generating new points from the current data, limiting the system's dependency on any individual data point [56]. However, by far the most common and effective method for solving overfitting is regularisation [109]. Most often overfitting occurs because the system can find a complex solution that exactly fits the data, rather than the simpler solution that would approximate the data within its error bound: this issue is also evidenced by Figure 2.7. If we can steer the system towards simpler solutions, then the overfitting issue will be resolved. This is achieved by modifying the supervised learning objective function from standard empirical risk to 'regularised empirical risk':

$$\arg\min_{\mathbf{w}\in\mathcal{W}} \frac{1}{k} \sum_{i=1}^{k} \ell\left(f\left(\mathbf{x_i}; \mathbf{w}\right), \mathbf{y_i}\right) + \lambda r\left(\mathbf{w}\right) \tag{2.3.1}$$

where $\lambda > 0$ is a balancing hyperparameter and r is a regulariser that returns high values for complex parameters. The flexibility of λ allows the complexity of a system

to be adjusted: if a supervised learning pipeline is consistently producing overly complicated models, then λ can be increased to impose harsher penalties and produce simpler models, and vice-versa. The choice of regulariser is much more interesting and problem dependent, but there are two very common choices that are effective in many settings:

1. LASSO regression uses $r = \|\cdot\|_1$, i.e. the 1-norm. This choice steers $\mathbf{w}$ towards values with few active features, i.e. many $\mathbf{w}^i = 0$. As such, it is effective when each parameter feature represents a distinct aspect of the system, e.g. monomials within polynomial regression [95].
2. Weight decay uses $r = \|\cdot\|_2^2$, i.e. the 2-norm squared. A less strict version of LASSO, this choice steers values $\mathbf{w}$ towards values with consistently low-valued features, i.e. $\left|\mathbf{w}^i\right| \approx \left|\mathbf{w}^j\right| \leq U$ for all i, j and some upper bound $U \in \mathbb{R}$. As such, it allows all parameters to be active, maximising system information, but still limits complexity and so is effective when the parameters are very dependent and interactive within the system, e.g. weights within a neural network [52].

Overall, regularisation is very common and so most supervised learning systems use the regularised empirical loss objective function.

We now examine some specific supervised learning models and their exact formulations.

Polynomial Fitting

In this situation, we wish to construct a predictive model and have determined that a polynomial function $f(\cdot, \mathbf{w})$ would be an appropriate model for the data $\mathcal{D}$. We can parametrise the class of polynomials up to order n like so:

$$\begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_n \end{bmatrix} \mapsto y = \beta_0 + \beta_1 x + \ldots + \beta_n x^n$$

Using this parametrisation, the parameter space is $\mathcal{W} = \mathbb{R}^n$. Note that we have implicitly assumed that the input and output spaces are one dimensional, i.e. $\mathcal{I}, \mathcal{O} \subseteq \mathbb{R}$. There are higher-dimensional variants of polynomial fitting [62], but their use is very uncommon.

The choice of loss function can vary, but the squared 2-norm is by far the most common:

$$\ell\left(f\left(x_i; \mathbf{w}\right), y_i\right) = \left(f\left(x_i; \mathbf{w}\right) - y_i\right)^2$$

When this 2-norm loss function is used, we get the 'Least-Squares' model for polynomial fitting. Finally, we must choose a regulariser. LASSO is often selected to encourage higher-order terms, i.e. β_j for j close to n, to be 0 and so not over-complicate the model. Overall, this gives us:

$$\arg\min_{\mathbf{w} \in \mathbb{R}^n} \frac{1}{k} \sum_{i=1}^{k} \left(\sum_{j=1}^{n} \mathbf{w}^j \left(x_i\right)^j - y_i \right)^2 + \lambda \|\mathbf{w}\|_1 \tag{2.3.2}$$

for some balancing hyperparameter $\lambda > 0$. Figure 2.7 was produced by running polynomial fitting models with and without LASSO regularisation.

Support Vector Machines

In this situation, we wish to construct a classifying model that distinguishes between two categories within the data. As such, the output space is simply $\mathcal{O} = \{-1, 1\}$, with -1 and 1 denoting the distinct categories. One method to distinguish between the categories is to find the maximum-margin hyperplane that separates the two groups such that the distance between the plane and any point $\mathbf{x_i}$ is maximised.

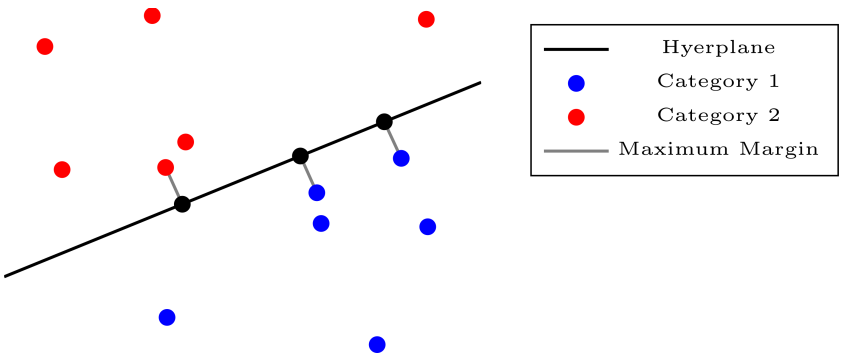


Figure 2.8: Maximum Margin Hyperplane

Any hyperplane can be defined by the equation

$$\langle \mathbf{w}, \mathbf{x} \rangle - b = 0$$

We want to ensure that all points from one category fall on one side, while all points from the other category fall on the other side, and ideally we want the points to be as far away from the separating hyperplane as possible. So we want:

$$\begin{aligned} \langle \mathbf{w}, \mathbf{x_i} \rangle - b \geq 1 \qquad & y_i = 1 \\ \langle \mathbf{w}, \mathbf{x_i} \rangle - b \leq -1 \qquad & y_i = -1 \end{aligned}$$

where b is minimum margin between the points and the hyperplane. We can put these constraints together into a single equation:

$$y_i \left(\langle \mathbf{w}, \mathbf{x_i} \rangle - b \right) \geq 1$$

for all $1 \leq i \leq k$. However, this constraint may be infeasible if the data is not linearly separable. In that case, we can aim to minimise the 'hinge-loss' for each data point:

$$\ell \left(f \left(\mathbf{x_i}; \mathbf{w} \right), y_i \right) = \max \left\{ 0, 1 - y_i \left(\langle \mathbf{w}, \mathbf{x_i} \rangle - b \right) \right\}$$

If $\mathbf{x_i}$ falls on the right side of the hyperplane, then the loss is 0, otherwise, the loss scales linearly with how far it is from the hyperplane on the wrong side. Finally, we want

to find the maximum margin hyperplane: if the points are linearly separable, then the margin is given by $\frac{b}{\|\mathbf{w}\|_2}$ and so we want to minimise $\|\mathbf{w}\|_2$. However, when the points are not linearly separable, then minimising this 2-norm will increase the loss for those points $\mathbf{x_i}$ on the wrong side of the hyperplane. So this norm becomes our regulariser, balancing our sampled data $\mathcal{D}$ and the overall model. Overall, we have the following optimisation problem for support vector machines:

$$\arg\min_{\mathbf{w}\in\mathbb{R}^n} \frac{1}{k}\sum_{i=1}^{k} \max\left\{0, 1 - y_i\left(\langle \mathbf{w}, \mathbf{x_i}\rangle - b\right)\right\} + \lambda \|\mathbf{w}\|_2 \tag{2.3.3}$$

for some balancing hyperparameter $\lambda > 0$.

Our current formulation of support vector machine only allows for linear separating boundaries between the two classes. For many data sets, this would not be sufficient for producing effective models. Fortunately, support vector machines can be extended to allow for non-linear boundaries. This is done using the 'kernel trick': the data is projected into a higher-dimensional space for which there is a separating hyperplane [4]. When this trick is used, the formulation can be simplified such that the only required change is that a kernel function $k : \mathcal{I} \times \mathcal{W} \to \mathbb{R}$ is used instead of the inner product $\langle \cdot, \cdot \rangle$ [92].

Neural Networks

Neural networks are built from collections of neurons, each performing simple calculations and passing that information downstream towards some output node. Neural networks have been the centre of phenomenal quantities of research within the last two decades, resulting in many different architectures and derived submodels being developed [64]. As such, it is near impossible to define the entire class of neural networks. Consequently, we will focus our attention on the original model, the multi-layered perceptron, upon which all other neural network models are built upon.

A multi-layered perceptron organises its neurons into layers. Each layer performs a matrix multiplication determined by the parameter $\mathbf{w}$, adds a bias vector, and then applies a final normalising/activation function to each component. These layers are connected as each previous layer's output is fed as the input into the successive layer. We can also take an individual neuron perspective. Each neuron performs an inner product with the output of the previous layer, adds its scalar bias, and then applies its activation function. This output is then used as one component in the vector output of the layer. From either perspective, we get the overall structure depicted in Figure A simple multi-layered perceptron with a single hidden layer of 3 nodes

The model is parametrised by the weights and biases used within each neuron, determining the other vector in the inner product and the bias added before the activation function. For example, the a-th component in a parameter $\mathbf{w}$ may determine the b-th component in the inner product vector for the c-th neuron in the d-th layer.

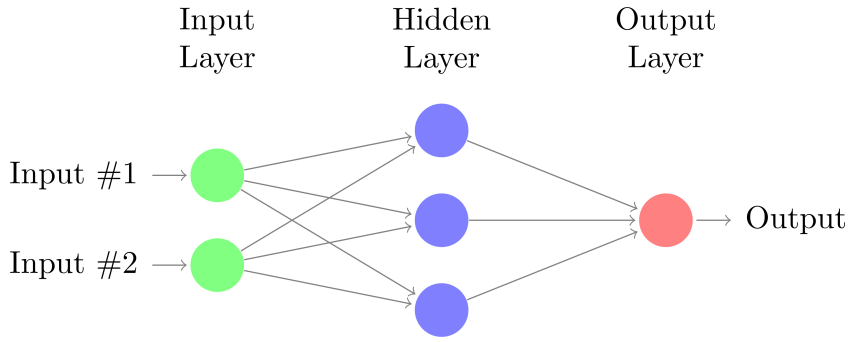


Figure 2.9: A simple multi-layered perceptron with a single hidden layer of 3 nodes

For our purposes, it is easiest to define a multi-layered perceptron with the following iterative relation:

$$\mathbf{o_i} = \sigma_i \left(V_i \left(\mathbf{w} \right) \mathbf{o_{i-1}} + \mathbf{b_i} \right) \tag{2.3.4}$$

for $i = 1, \ldots, L$, where $L \in \mathbb{N}$ is the number of layers in the network, $\mathbf{o_i}$ is the output of the i-th layer, $V_i(\cdot)$ are linear maps from the parameter space $\mathcal{W}$ into the space of matrices, i.e. $V_i(\mathbf{w})$ is a matrix for each $\mathbf{w} \in \mathcal{W}$, $\mathbf{b_i}$ is an added bias vector, and σ_i is the component-wise activation function for layer i. Note that the number of neurons in each layer can vary, and usually does, vary between layers. Finally, using the input data as the original input $\mathbf{o_0} = \mathbf{x_i}$, we get:

$$f\left(\mathbf{x_i}; \mathbf{w} \right) = \mathbf{o_L} \tag{2.3.5}$$

i.e. the output of the final layer is the output of the modelling function.

There are many different activation functions: the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ was the initial activation function used, but modern networks tend to use other activations functions, such as the 'Rectified Linear Unit' $\mathrm{ReLU}(x) = \max\{0, x\}$ and the hyperbolic tangent function $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, as they have shown better performance [81].

While multi-layered perceptrons may seem interesting, they are certainly far more complicated than most other models and require significant computation for even a few layers. As such, motivation is required for their use. This comes in the form of the Universal Approximation Theorem: roughly speaking, any continuous function can be approximated within any accuracy by a multi-layered perceptron [33]. This arbitrary accuracy potentially requires infinitely large networks (either an infinite number of layers or an infinite number of neurons in a layer) which is practically impossible, but finite accuracy is still achievable with finite networks [32]. This is an incredibly strong result: multi-layered perceptrons can be used to approximate most relations.

This result motivates the use of multi-layered perceptrons. Many unknown relations are sufficiently complex that exploratory statistical analysis will not help determine what class of functions $\{f(\cdot; \mathbf{w})\}_{\mathbf{w} \in \mathcal{W}}$ would produce reasonable approximations. In these cases, a multi-layered perceptron model can be used as any non-pathological relation can

be well approximated by a sufficiently large network. Consequently, many complicated relations and systems are modelled using multi-layered perceptrons, making them one of the most widely used machine learning models [3].

Given the success of multi-layered perceptron, it is very clear why the large family of neural networks have been built from them. In fact, the scope of this entire family and its applications is so wide, that little can be said in general about their supervised learning pipeline: many different loss functions are used, from simple 2-norms to softmax entropy functions; the data varies from discrete and one-dimensional to continuous and multi-dimensional; a variety of regularisers or regularising techniques are applied, from simple LASSO to neuron drop-out methods; and, in the end, the networks are used for prediction, classification, model insight, and more.

Deep Declarative Networks

Deep declarative networks are a form of bi-level optimisation model that extend the class of neural networks [47]. This extension is made allowing for 'declarative' nodes within the network. The forward function of these nodes is not given by an inner product, additive bias, and activation function, but is instead defined as a parametrised optimisation problem:

$$\mathbf{o_i}^j \in \underset{\mathbf{x}\in\mathcal{C}(\mathbf{o_{i-1}};\mathbf{w})}{\arg\min}\ f\left(\mathbf{x}, \mathbf{o_{i-1}}; \mathbf{w}\right) \tag{2.3.6}$$

i.e. the j-th component of the output from the i-th layer is given by an optimisation problem whose constraint set and objective function are parametrised by the output of the $i-1$-th layer and the parameter $\mathbf{w}$. For example, a layer of declarative nodes could be set up such that the layer's output is a L_p-ball projection:

$$\mathbf{o_i} = \underset{\mathbf{x}\in\left\{u\in\mathbb{R}^n:\|u\|_p=1\right\}}{\arg\min}\ \frac{1}{2}\left\|\mathbf{x}-\mathbf{o_{i-1}}\right\|_2^2$$

for some $p \in \mathbb{N}$. These declarative nodes can co-exist with standard 'imperative' neurons within one network.

Given that standard neural networks can already approximate a large class of functions, we must ask why deep declarative networks are necessary. The class of functions is not extended by deep declarative networks, however much more control is granted over each network. All standard neural networks begin with no prior towards the data and relation: the training procedure must move the parameters such that the final model is a good approximation. In contrast, specific declarative nodes can be set up with a prior towards the relation and data: declarative nodes can be defined that produce targeted outputs, such as robust feature pooling or projections to specific or parametrised sets. As such, deep declarative networks do not only rely upon the training process but incorporate a structural prior for the problem. While this does introduce structural complexities that can potentially slow the inference time, the structure prior can reduce training time, improve robustness, and increase model explainability and interpretability [47].

The optimisation problems related to the declarative nodes are called the 'forward' and 'low-level' optimisation problems, whereas the overall optimisation problem related to the regularised empirical loss is called the 'backward' and 'high-level' optimisation problem. This nomenclature highlights that deep declarative networks are a form of 'bi-level' optimisation [40]. This optimisation and learning paradigm has shown increased interest over the past few years as many different problems, such as hyperparameter optimisation and adversarial learning [63], can be formulated as bi-level problems. As such, there is ongoing research into efficient and effective algorithms for optimising both the low-level and high-level problems [16].

2.3.3 Classification of Machine Learning Optimisation Problems

We now quickly go back over the various machine learning systems that we have considered and classify their optimisation problems.

For the general supervised machine learning problem, as defined by Equation (2.3.1), not much can be said. This system is constructed to learn any possible relation $P(\mathbf{x}, \mathbf{y})$ using any parametrised class of functions $\{f(\cdot; \mathbf{x})\}_{\mathbf{x} \in \mathcal{W}}$ and so there are too many unknowns to concretely classify the problem completely.

That said, two qualities are constant across all supervised machine learning problems. Firstly, supervised machine learning problems are global optimisation problems: the domain is not restricted to convex problems, but instead, the entire domain of the functions is considered. Secondly, supervised machine learning problems are stochastic. This stochastic quality is introduced by the sampling process of the relation $P(\mathbf{x}, \mathbf{y})$ to produce the data $\mathcal{D}$: what data is used to represent the relation is entirely random and so the problem is stochastic. Furthermore, even if the data could represent the entire distribution accurately, which is sometimes possible for discrete distributions, then we encounter the issue that most sampling methods are themselves stochastic, due to human, measurement, or computational error. As such, supervised learning problems are global, stochastic optimisation problems. This is the limit of the classification for general supervised machine learning problems. To be more precise, we must consider specific systems and models.

The four specific models we considered - polynomial fitting models (2.3.2), support vector machines (2.3.3), multi-layered perceptrons (2.3.4) and (2.3.5), and deep declarative networks (2.3.6) - are all global, unconstrained, continuous, non-smooth, stochastic optimisation problems. The unconstrained and continuous qualities arise immediately from $\mathcal{W} = \mathbb{R}^n = \operatorname{dom}(f)$, even if the exact use of the parameter is different in each case. The non-smooth quality generally arises due to the regulariser: LASSO $\|\cdot\|_1$, weight-decay $\|\cdot\|_2$, and most multi-layered perceptron regularisers, such as neuron drop-out, are all non-smooth with numerous non-differentiable points. However, aside from polynomial fitting, even without regularisation, the max function in the hinge-loss for support vector machines, ReLU and other activation functions in multi-layered perceptrons, and many

of the low-level forward optimisation problems in deep declarative networks would make the objective function non-smooth anyway.

So all machine learning systems that we considered are global, unconstrained, continuous, non-smooth, stochastic optimisation problems. As noted previously, despite limiting our attention to supervised learning models and only a small handful of models, the group is representative of the wider machine learning field: except for some clustering algorithms such as hierarchical clustering that are discrete optimisation problems [70], the vast majority of machine learning problems fall into this form of optimisation problem. This fact, in addition to the pervasiveness and significance of machine learning, is our motivation for conducting research into such optimisation problems and their algorithms.

Chapter 3

Stochastic Subgradient Descent

This chapter is dedicated to the theory of Stochastic Subgradient Descent. We begin by deriving the algorithm as a generalisation of Stochastic Gradient Descent. We highlight that its convergence is non-trivial, not immediately extending from the Convergence of Stochastic Gradient Descent. As a result, we dedicate time to presenting the convergence result, outlining the methods and concepts that form the proof. Finally, we discuss the algorithm's theory and consider various extensions and applications, particular to machine learning.

3.1 Non-Smooth Optimisation

In this section, we outline the primary problem that we are facing: namely, standard smooth optimisation methods are not sufficient for training machine learning models due to their non-smooth components. We then show how this problem can be solved by generalising the techniques used in Smooth Optimisation to the non-smooth domain. We develop this generalised theory and then present the Stochastic Subgradient Descent algorithm. We then demonstrate that while much of the practice surrounding Stochastic Gradient Descent can be extended to Stochastic Subgradient Descent, the analysis, particularly Convergence of Stochastic Gradient Descent, does not follow this pattern.

3.1.1 Applying Smooth Optimisation Techniques to Machine Learning

The training process is the primary step in developing and producing machine learning models: architectures can be designed, data collected and processed, but if the model does not learn, then everything is for naught. Fortunately, as illustrated in Machine Learning, the vast majority of models are simply specific optimisation problems and so we can train them by applying optimisation methods.

Given their ubiquity and also the depth of the existing theory, smooth optimisation methods were and still are used to train machine learning models [94]. However, there is a critical complication with this approach. As we highlighted in Classification of Machine Learning Optimisation Problems, most models incorporate non-smooth components, and so their objective functions are not smooth and can not be differentiated everywhere. Consequently, applying smooth optimisation methods to these non-smooth problems causes a raft of significant issues.

Firstly, there is no longer a guarantee of convergence. The pre-conditions for the convergence results of smooth methods require that the function be differentiable, sometimes even continuously differentiable as is the case in Convergence of Stochastic Gradient Descent. So smooth optimisation techniques can not be applied to machine learning with a guarantee of convergence. In practice, this could lead to situations in which the model does not or is even incapable of settling to an optimal point. At best, this would increases model instability and lowers robustness [29]; at worst, the model might learn nothing and produce non-sensical outputs.

Secondly, smooth methods may 'crash' if applied to a non-smooth objective function: if the objective function f is not differentiable at an iterate $\mathbf{x_i} \in \mathbb{R}^d$, then $\nabla f(\mathbf{x_i})$ is undefined and the algorithm is unable to complete the step and calculate the successive iterate $\mathbf{x_{i+1}}$: Figure 3.1 illustrates an example of an undefined gradient that would 'crash' a smooth optimisation algorithm.

Given that machine learning models have non-differentiable points, smooth optimisation methods may simply 'crash' if used for training, outputting a model of undetermined quality. This would be a disastrous result, wasting computational resources and halting application development. We note that modern frameworks for machine learning, such as PyTorch and TensorFlow [75, 2], would not 'crash' but rather use an incorrect gradient to compute the next iteration. This is certainly better than 'crashing', but it exacerbates the performance and robustness issues.

So applying smooth optimisation techniques to machine learning models causes a raft of issues, most critically by eliminating any guarantees on the robustness and performance of the model. However, these techniques are the only methods with the required knowledge and expertise for general application, forcing practitioners to accept subpar models[1]. This situation needs a resolution.

A potential solution to these issues is to approximate the derivative using numerical methods, such as forward or backward differences:

$$\frac{\partial f}{\partial \mathbf{x}^i} \approx \frac{f(\mathbf{x} + t\mathbf{e_i}) - f(\mathbf{x})}{t} \qquad \text{for some small } t > 0$$

However, these approximations carry their own issue: they can be very volatile, amplifying errors from noise, and so often are not accurate enough for gradient descent

[1] The situation begs the question how it has continued for so long. This is resolved in our discussion of the Computability of the Clarke Subdifferential.

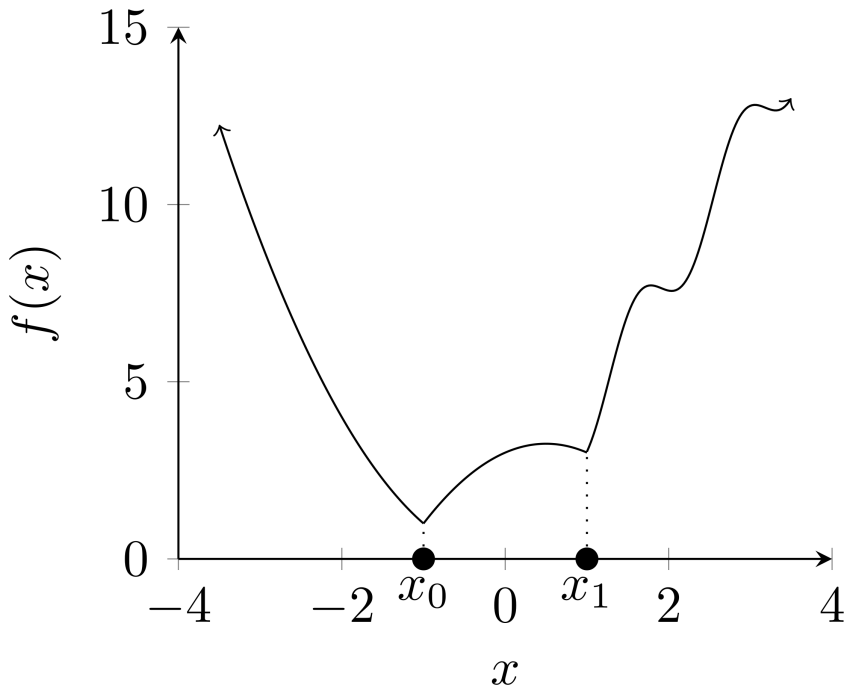


Figure 3.1: $f(x) = \begin{cases} x^2 & x \leq -1 \\ 3 + x - x^2 & -1 \leq x \leq 1 \\ 4x + \sin(\pi x) & x \geq 1 \end{cases}$ has undefined gradients at $x = -1, 1$.

methods in stochastic domains [48, 28]. In turn, this volatility issue can be overcome by regularising the differentiating process [59]. But such regularisation further removes the approximation from the mathematical definition of the gradient, making it difficult to guarantee convergence or even reason about the algorithm's behaviour. Consequently, using numerical methods to approximate the derivative is problematic for applying smooth optimisation methods to machine learning models.

Fortunately, there is a clear solution: if machine learning training is a non-smooth problem, then we simply ought to use non-smooth optimisation techniques.

3.1.2 Generalising Stochastic Gradient Descent

Using non-smooth techniques may seem obvious, but it does have a significant disadvantage: the well-established theory and expertise of the smooth optimisation methods, in particular the intuition of gradient descent methods, is forgone. To mitigate this disadvantage, non-smooth optimisation is framed and then developed as a generalisation of smooth optimisation: we derive generalised variants of the gradient and derivative, and then apply these variants within descent methods to develop non-smooth algorithms.

Clarke Subdifferential and Directional Derivative

We consider a particular class of functions:

Definition 3.1 (Locally Lipschitz Functions)**.** *Let $f : X \subseteq \mathbb{R}^d \to \mathbb{R}$ be a function. It is* Locally Lipschitz *if on all bounded sets $B \subseteq \mathbb{R}^d$, there exists a constant L_B such that*

$$|f(\mathbf{x}) - f(\mathbf{y})| \leq L_B \|\mathbf{x} - \mathbf{y}\|_2$$

for all $\mathbf{x}, \mathbf{y} \in B$, i.e. on any bounded set in $\mathbb{R}^n$, f is limited by how steep it can get within that set.

This class of functions is very broad, encompassing all the machine learning optimisation problems outlined so far and the vast majority of others - any exceptions would be isolated cases. Thus, if we can derive a generalised derivative and gradient for this class, then we can apply it to the machine learning domain. Furthermore, (local) Lipschitz continuity is a standard condition for convergence of optimisation algorithms, recall the assumption Lipschitz-Continuity of Objective Gradient, and so deriving a generalised derivative for this class of functions will not limit wider application.

A well-known result of continuity shows that such functions are differentiable almost everywhere:

Theorem 3.1 (Rademacher's Theorem)**.** *Let $f : X \subseteq \mathbb{R}^d \to \mathbb{R}$ be a function and $U \subseteq X$ an open subset. If $f\,|_U$, i.e. f restricted to U, is Lipschitz continuous, then f is differentiable almost everywhere[a] on U. In particular, locally Lipschitz functions are differentiable almost everywhere.*

[a] Almost everywhere is a technical term from measure theory that can be roughly interpreted as there is 100% probability that a randomly selected point satisfies the condition, even if points exist that do not satisfy the condition. A simple example is that the real numbers $\mathbb{R}$ are not integers $\mathbb{Z}$ almost everywhere.

Given this result, we can construct a generalised derivative:

Definition 3.2 (Clarke Subdifferential)**.** *Suppose $f : X \subseteq \mathbb{R}^d \to \mathbb{R}$ is a locally Lipschitz function. The* Clarke Subdifferential *of f at any point $\mathbf{x} \in X$ is given by:*

$$\partial f(\mathbf{x}) = \operatorname{conv}\left\{\lim_{i\to\infty} \nabla f(\mathbf{x_i}) : \mathbf{x_i} \overset{\Omega}{\to} \mathbf{x}\right\}$$

where Ω is any full-measure[a] subset of X on which f is differentiable and conv *denotes the convex-hull operator[b].*

[a] Another technical term from measure theory that roughly means: the set is almost everywhere

within its ambient superset, e.g. the irrational numbers are a full-measure set in $\mathbb{R}$.
[b]The convex-hull of a set A is the minimal superset $B \supseteq A$ such that B is convex.

This definition is rather complicated: the Clarke subdifferential $\partial f(\mathbf{x})$ is given by the convex hull of the gradients of the sequences limiting to $\mathbf{x}$. Fortunately, we can visualise the construction in some cases: Figure 3.2 illustrates such an example.

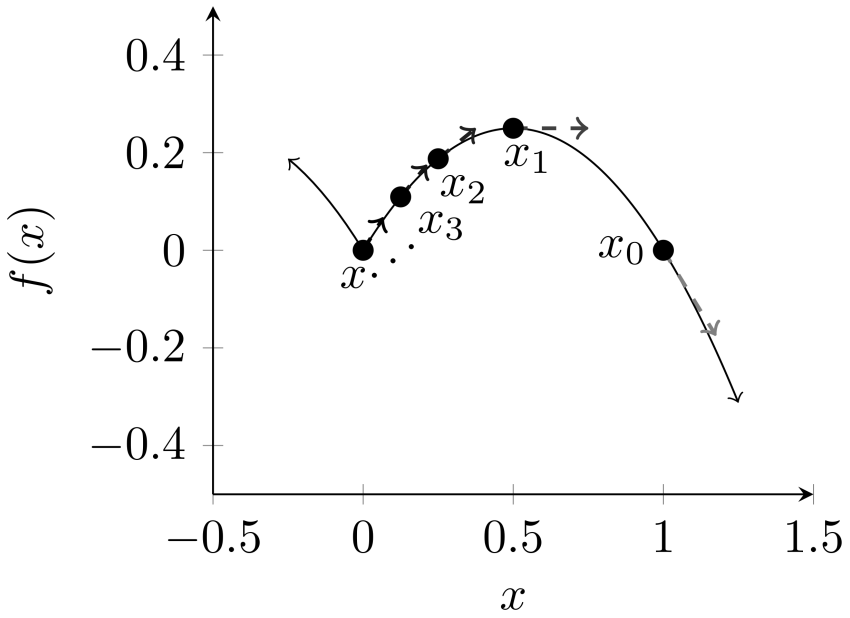


Figure 3.2: Clarke Subdifferential Construction: Given the sequence $\{x_i\}_{i\geq 0}$ which converges to x, we can take the limit $\lim_{i\to\infty} \nabla f(x_i)$ to produce one value in the Clarke Subdifferential. If we repeat this process for all sequences $x_i \to x$ and then take the convex hull of the results, we produce the entire Clarke Subdifferential.

However, visualisations are not always available, so it is good to have a grasp on the basic properties:

Proposition 3.1 (Properties of the Clarke Subdifferential)**.** *([30] Sec. 1) Let $f : X \subseteq \mathbb{R}^d \to \mathbb{R}$ be locally Lipschitz. Let $\mathbf{x} \in X$. The following are true:*

1. *$\partial f(\mathbf{x})$ is a non-empty, convex, compact set.*
2. *The operator $\partial f : X \rightrightarrows^{a} \mathbb{R}^d$ is outer-semicontinuous: if $\mathbf{v_i} \to \mathbf{v} \in \mathbb{R}^d$, i.e. the sequence $\{\mathbf{v_i}\}_{i\geq 0}$ converges to $\mathbf{v} \in \mathbb{R}^d$, and $\mathbf{x_i} \to \mathbf{x}$, and if each $\mathbf{v_i} \in \partial f(\mathbf{x_i})$, then $\mathbf{v} \in \partial f(\mathbf{x})$.*

Furthermore, the following are equivalent:

1. $\partial f(\mathbf{x}) = \{\xi\}$ *is a singleton.*
2. $\nabla f(\mathbf{x})$ *exists,* $\nabla f(\mathbf{x}) = \xi$*, and* ∇f *is continuous at* $\mathbf{x}$ *relative to* X*.*

[a]Recall that this is the notation for a set mapping, i.e. f maps from X to subsets of $\mathbb{R}^d$.

The first pair of properties relate to the nature of the Clarke subdifferential operator, $\partial f : X \rightrightarrows \mathbb{R}^d$, and its outputs. The second pair of properties specify the relation between the Clarke subdifferential and the standard gradient: the Clarke subdifferential is an extension of the gradient, reducing to the gradient at differentiable points. This means that any Clarke subdifferential descent method will reduce to a standard gradient descent method on smooth functions.

Naturally, these properties apply to all locally Lipschitz functions, even those that seem strange or pathological. Such a case may be the following: We can define infinitely many piecewise linear functions

$$\begin{aligned} f_n &: \mathbb{R} \to \mathbb{R} \\ x &\mapsto \begin{cases} \frac{3}{2^n}x - 2^{1-2n} & x \in \left[\frac{1}{2^n}, \frac{1}{2^{n-1}}\right] \\ 3 \times 2^n x - 2^{2n+1} & x \in \left[2^n, 2^{n+1}\right] \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{3.1.1}$$

such that their sum is a continuous piece-wise linear function:

$$\begin{aligned} f &: \mathbb{R} \to \mathbb{R} \\ x &\mapsto \sum_{i \in \mathbb{N}} f_i(x) \end{aligned} \tag{3.1.2}$$

This function f has the interesting property that it is non-differentiable at infinitely many points $\left\{\frac{1}{2^n}\right\}_{n\in\mathbb{N}}$. Yet f is locally Lipschitz, as its gradient is linear within any local neighbourhood, so it is still differentiable almost everywhere. As such, $\partial f(x)$ exists for all $x \in [-1, 1]$ and satisfies all the basic properties. In particular, despite there being infinitely many non-differentiable points around 0, we can still calculate that $0 \in \partial f(0)$ by applying the outer-semicontinuity property of the Clarke subdifferential on the sequence of differentiable points $\left\{-\frac{1}{2^n}\right\}_{n\in\mathbb{N}}$. In fact, $\partial f(0) = \{0\}$ as the gradients of any sequence of points limiting to 0 tend towards 0, even when approaching from the right.

This example also evidences some awkwardness with the Clarke subdifferential: easy intuition is lost in the various complexities of convex hulls and limit points. Fortunately, we can derive an equivalent definition that is much more intuitive. We do this by extending the directional derivative:

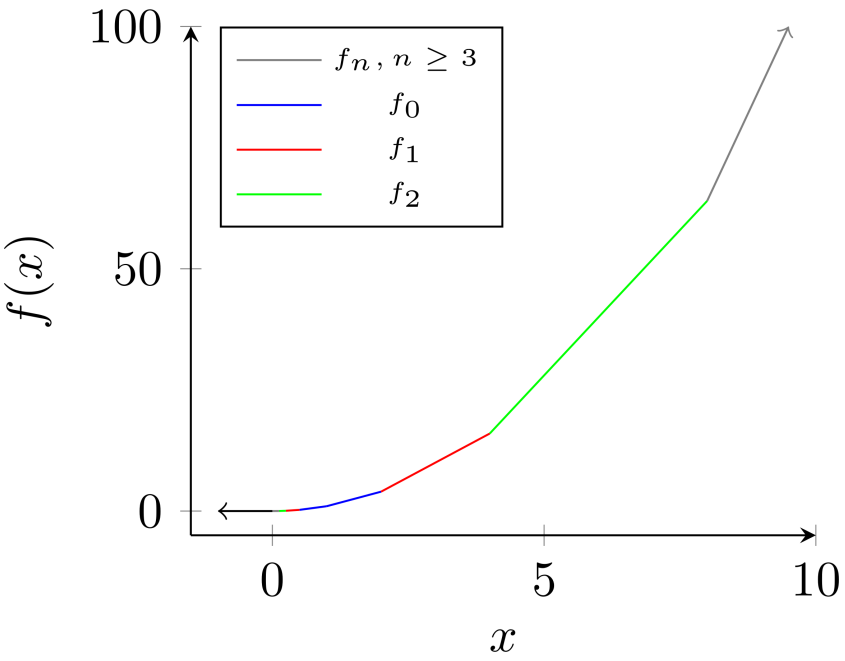


Figure 3.3: f_n and f

Definition 3.3 (Directional Derivative)**.** *Suppose $f : X \subseteq \mathbb{R}^d \to \mathbb{R}$ is a function. Then the* Directional Derivative *of f at $\mathbf{x} \in X$ in direction $\mathbf{v} \in \mathbb{R}^d$, with $\mathbf{v} \neq \mathbf{0}$, is given by:*

$$f'(\mathbf{x}; \mathbf{v}) = \lim_{t \searrow 0} \frac{f(\mathbf{x} + t\mathbf{v}) - f(\mathbf{x})}{t}$$

provided the limit exists.

When f is continuous, this limit will exist and so all locally Lipschitz functions have well-defined directional derivatives at all points in their domains. We now generalise this idea in a manner similar to the Clarke subdifferential:

Definition 3.4 (Clarke Directional Derivative)**.** *Suppose $f : X \subseteq \mathbb{R}^d \to \mathbb{R}$ is a locally Lipschitz function. The* Clarke Directional Derivative *of f at any point $\mathbf{x} \in X$ in direction $\mathbf{v} \in \mathbb{R}^d$ is given by:*

$$f^\circ(\mathbf{x}; \mathbf{v}) = \lim_{\mathbf{y} \to \mathbf{x}} \sup_{t \searrow 0} \frac{f(\mathbf{y} + t\mathbf{v}) - f(\mathbf{y})}{t}$$

So this construction extends the standard directional derivative by considering the supremum of directional derivatives for sequences $\mathbf{y} \to \mathbf{x}$. As such, the Clarke directional derivative at $\mathbf{x}$ in some direction $\mathbf{v}$ can be loosely interpreted as largest standard directional derivative in direction $\mathbf{v}$ at any point within the local neighbourhood of $\mathbf{x}$.

These supremum and local neighbourhood aspects produce some interesting results. For example, for the function

$$\begin{aligned} f : \mathbb{R} &\to \mathbb{R} \\ x &\mapsto -|x| \end{aligned}$$

the Clarke directional derivatives at $x = 0$ are given by

$$f^{\circ}(0; v) = |v|$$

for all $v \in \mathbb{R}$. So even though the standard directional derivative 0 is always negative, the Clarke directional derivative is always positive. This is because we can always approach $x = 0$ with a sequence from the left side and so produce a positive slope.

We can bring over our basic intuitions for the Clarke directional derivative to the Clarke subdifferential with the following relation:

Theorem 3.2 (Clarke Subdifferential by Directional Derivatives)**.** *([31], Chp. 2, Sec. 8, Thm. 8.1) Suppose $f : X \subseteq \mathbb{R}^d \to \mathbb{R}$ is locally Lipschitz and $\mathbf{x} \in X$. Then we have that:*

$$\partial f(\mathbf{x}) = \left\{ \xi \in \mathbb{R}^d : f^{\circ}(\mathbf{x}; \mathbf{v}) \geq \langle \xi, \mathbf{v} \rangle, \forall \mathbf{v} \in \mathbb{R}^d, \mathbf{v} \neq \mathbf{0} \right\}$$

i.e. the Clarke subdifferential is given by the set of vectors such that their inner product with a direction is less than the Clarke subdifferential in that same direction, for all directions.

Recalling our previous interpretation of the Clarke directional derivative and the relation $f'(\mathbf{x}; \mathbf{v}) = \langle \nabla f(\mathbf{x}), \mathbf{v} \rangle$ when f is differentiable at $\mathbf{x}$, we can now interpret the Clarke subdifferential $\partial f(\mathbf{x})$ as the set of alternative gradients ξ that produce smaller directional derivatives values than the largest directional derivative value in the local neighbourhood of $\mathbf{x}$.

While this interpretation is potentially a bit more concrete than the standard definition, it is far from intuitive. Fortunately, in a specific but relevant setting, we can strengthen the link between the standard and Clarke directional derivatives and yield a better understanding of the Clarke subdifferential:

Definition 3.5 (Regular Functions)**.** *([9], Chp. 6, Sec. 4, Def. 6.5) Let $f : X \subseteq \mathbb{R}^d \to \mathbb{R}$ be Lipschitz near $\mathbf{x} \in \mathbb{R}^d$. The function f is said to be* Regular *at $\mathbf{x}$ if*

$$\forall \mathbf{v} \in \mathbb{R}^d, f'(\mathbf{x}; \mathbf{v}) = f^{\circ}(\mathbf{x}; \mathbf{v})$$

i.e. the standard directional derivative and Clarke directional derivative agree in all directions.

Proposition 3.2 (Lipschitz and Convexity Imply Regularity)**.** *([9], Chp. 6, Sec. 4, Thm. 6.8) Let $f : X \subseteq \mathbb{R}^d \to \mathbb{R}$. If f is Lipschitz and convex near $\mathbf{x} \in X$, then f is regular at $\mathbf{x}$.*

So, if f is locally Lipschitz and convex near $\mathbf{x}$, then the Clarke directional derivative is equivalent to the standard directional derivative. Fortunately, we are already assuming that f is locally Lipschitz and optimal points tend to have convex neighbourhoods, the only exceptions being pathological examples such as the Weierstrass function[2], so this equivalence generally holds once we are close enough to an optimal point. Consequently, we can build a good intuition for the Clarke subdifferential in such regions: If f is regular at $\mathbf{x} \in X$, then we have that

$$\partial f(\mathbf{x}) = \left\{ \xi \in \mathbb{R}^d : f'(\mathbf{x}; \mathbf{v}) \geq \langle \xi, \mathbf{v} \rangle, \forall \mathbf{v} \in \mathbb{R}^d, \mathbf{v} \neq 0 \right\}$$

Then, again recalling that $f'(\mathbf{x}; \mathbf{v}) = \langle \nabla f(\mathbf{x}), \mathbf{v} \rangle$ when f is differentiable at $\mathbf{x}$, we can interpret the values ξ as alternative gradients that produce smaller alternative directional derivative values than the standard directional derivative. As such, the set of Clarke subdifferentials can be, in most of our cases, intuitively understood to be gradients of the tangent lines or planes that remain beneath the function within the local region. Figure 3.4 illustrates this concept.

Other Generalised Derivatives

We quickly comment on the existence and use of other generalised derivatives.

Firstly, the Clarke Subdifferential can be extended to functions that are not locally Lipschitz, but merely bounded [30]. However, this extension is rather complex, involving tangent and normal cones, and many basic properties, such as $\partial f(\mathbf{x})$ always being non-empty, are lost. Additionally, it is simply not required for our purposes. So we leave it as further reading for any interested party.

Secondly, there are many generalised derivatives and directional derivatives distinct from the Clarke subdifferential and directional derivative. Ralph Rockafellar and Roger Wets devote an entire chapter in their seminal text 'Variational Analysis' [82] to the concept, outlining various alternative constructions and examining their relations.

This prompts the question of why the Clarke subdifferential is significant. Rockafellar and Wets also answer this question, showing that Clarke regularity ([82] Chp. 7, Sec. D, Def. 25) implies a duality between subdifferentials and directional derivatives: specifically, Clarke regularity extends the duality between gradient vectors and linear functions to a bijective correspondence between a closed convex set of vectors and sublinear functions that support a half-space, i.e. Clarke Subdifferential by Directional Derivatives. This correspondence explains the significance of the Clarke subdifferential

[2] See Limits of Assumptions for an analysis of the Clarke Subdifferential of the Weierstrass function.

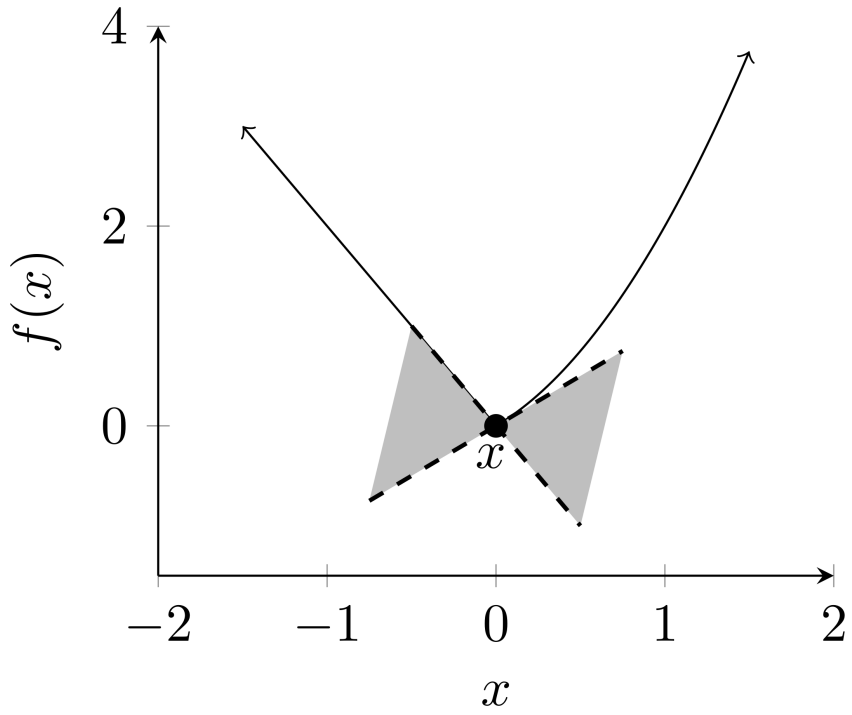


Figure 3.4: Clarke Directional Derivative Interpretation of Clarke Subdifferential: Any line through x that remains in gray area will remain beneath the function within the local region. The gradients of these lines form the Clarke Subdifferential $\partial f(0)$.

and its subsequent application in optimisation. That said, alternative algorithms that use different generalised derivatives exist for specific settings. For example, Fénchel subgradient methods, such as the Douglas-Rachford splitting method [37] or AdaGrad [38], are commonly used in convex settings.

Non-Smooth Optimality Conditions

In parallel to the results derived in Smooth Optimality Conditions, we now outline the optimality conditions that relate to the Clarke subdifferential. We begin with the generalisation of a stationary point:

Definition 3.6 (Clarke Critical Points and Values)**.** *Suppose* $f : X \subseteq \mathbb{R}^d \to \mathbb{R}$ *is a locally Lipschitz function. Then* $\mathbf{x} \in X$ *is a* Clarke Critical Point *of* f *if* $0 \in \partial f(\mathbf{x})$*. A point* $r \in \mathbb{R}$ *is a* Clarke Critical Value *of* f *if there exists a critical point* $\mathbf{x} \in \mathbb{R}^d$ *of* f *such that* $f(\mathbf{x}) = r$*.*

Note that for a point to be critical, the Clarke subdifferential does not need to be the singleton $\{0\}$, but rather just include 0, In fact, Properties of the Clarke Subdifferential

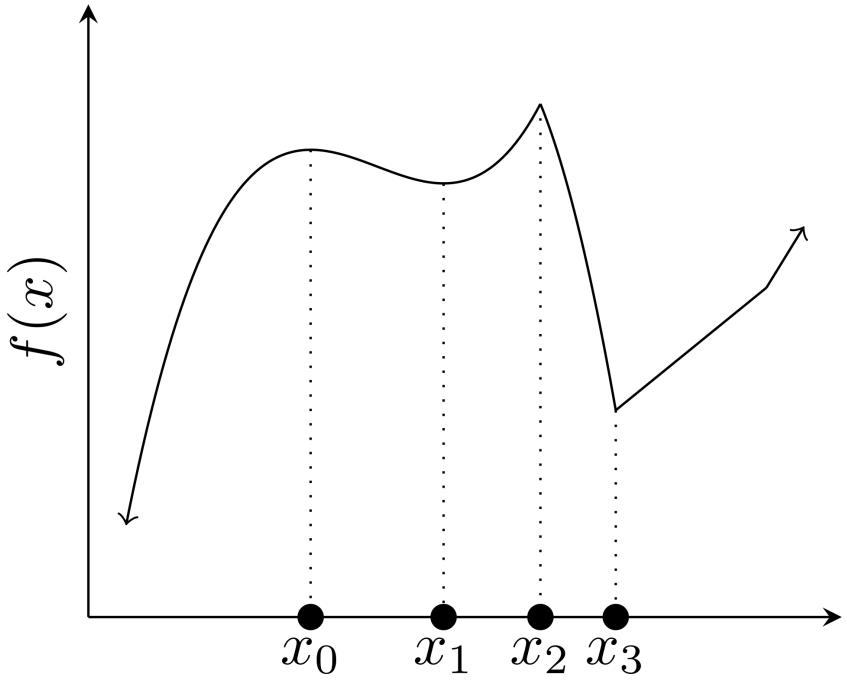


Figure 3.5: Clarke Critical Points: f is continuous for $x < x_2$ and so the Clarke Critical points are the Fermat Stationary points with gradients of 0. Using the Clarke Subdifferential by Directional Derivatives Theorem, we can determine that $0 \in \partial f(x_2), \partial f(x_3)$ as well.

implies that if $\partial f(\mathbf{x}) = \{0\}$, then f is differentiable at $\mathbf{x}$ and so the point is also a stationary point. As such, Clarke critical points are extensions of stationary points.

There is an equivalent representation of Clarke critical points using the Clarke directional derivative:

Theorem 3.3 (Clarke Critical Points by Directional Derivatives)**.** *Suppose $f : X \subseteq \mathbb{R}^d \to \mathbb{R}$ is a locally Lipschitz function. A point $\mathbf{x} \in X$ is a Clarke critical point if and only if*

$$f^\circ(\mathbf{x}; \mathbf{v}) \geq 0 \qquad \forall \mathbf{v} \in \mathbb{R}^d, \mathbf{v} \neq 0$$

i.e. the Clarke directional derivative is non-negative in all directions.

Using the Clarke directional derivative and considering the case of f being regular, i.e. also locally convex, once again makes the intuition regarding Clarke critical points clear: they are points at which any direction is uphill. In this light, we can derive the generalisation of Fermat's Stationary Point Theorem:

Theorem 3.4 (Critical Point Theorem)**.** *([82], Chp. 10, Sec. A, Thm. 1)Let $f : X \subseteq \mathbb{R}^d \to \mathbb{R}$ be a locally Lipschitz function. If $\mathbf{x}$ is a minimum or maximum of f, then $\mathbf{x}$ is a Clarke critical point of f.*

Once again, the contrapositive yields the primary optimality condition:

Proposition 3.3 (Optimality Condition for Non-Smooth Optimisation)**.** *Suppose we have an optimisation problem of the form (2.1.2) where f is locally Lipschitz and let $\mathbf{x} \in \mathbb{R}^d$. If $\mathbf{x}$ is not a Clarke critical point, then $\mathbf{x}$ is not an optimal point.*

Again, we emphasise that this optimality condition only applies to our particular flavour of global unconstrained, continuous, non-smooth optimisation problems: the proof in Ralph Rockafellar and Roger Wets's 'Variational Analyis' outlines how it will fail if the conditions are not satisfied [82]. Such a strong result does not apply in more general settings.

In the smooth domain, we also derived a Second-Order Optimality Condition for Smooth Optimisation. This was done to eliminate maxima and achieve an explicit representation of the optimal points, see Equation (2.2.2). We can not derive such a result for non-smooth optimisation, as we can not iterate the Clarke subdifferential on itself given that it is a set-valued function. However, as we intend to emulate gradient descent methods with the Clarke subdifferential, if we can verify that an algorithm moves the iterates downhill then we will naturally avoid maxima.

Stochastic Subgradient Descent

We now have the theoretical tools to develop a generalised gradient descent method using non-smooth techniques. We recall that descent methods are a group of optimisation methods that aim to improve the current solution by moving downhill in each successive iteration: Figure 2.5 illustrates this concept. Previously, we used the gradient for such a descent algorithm: Stochastic Gradient Descent. That algorithm has the following primary loop for producing iterates:

$$\mathbf{x_{k+1}} = \mathbf{x_k} - \alpha_k \left(\nabla f \left(\mathbf{x_k} \right) + \xi_\mathbf{k} \right)$$

where $\{\alpha_k\}_{k \geq 0}$ is the sequence of step sizes and $\{\xi_\mathbf{k}\}_{k \geq 0}$ is the sequence of the gradient's random noise, sampled from some distribution. Given that the Clarke subdifferential is an extension of the gradient, we can easily generalise this algorithm to non-smooth functions by exchanging the gradient $\nabla f \left(\mathbf{x_k} \right)$ with a $\mathbf{g_k} \in \partial f \left(\mathbf{x_k} \right)$:

We quickly note that this algorithm is called stochastic *subgradient* descent because the elements of the Clarke Subdifferential are sometimes referred to as Clarke subgradients. However, this nomenclature is not used consistently and so we avoid its use everywhere except here.

Algorithm 4: Stochastic Subgradient Descent

input : Optimisation problem of form (2.1.2) where f is locally Lipschitz. Initial point $\mathbf{x_0} \in \mathbb{R}^d$. Step limit $N \in \mathbb{N}$. Sequence of step sizes $\{\alpha_k\}_{k=0}^{N} \in \mathbb{R}^N$.

output: Approximate locally optimal point $\mathbf{x}^* \in \mathbb{R}^d$ and approximate locally optimal value $p^* \in \mathbb{R}$.

Repeat iterative updates;

for $k = 1, 2, \ldots, N$ **do**

  Sample an approximate subdifferential;

  $\mathbf{g_k} \leftarrow \partial f(\mathbf{x_k}) + \xi_k$;

  Set new iterate;

  $\mathbf{x_{k+1}} \leftarrow \mathbf{x_k} - \alpha_k \mathbf{g_k}$;

Return locally optimal point and value;

$\mathbf{x}^* \leftarrow \mathbf{x_N}, p^* \leftarrow f(\mathbf{x_N})$;

return $\mathbf{x}^*, p^*$;

We also highlight that the sampling process for $\mathbf{g_k} \in \partial f(\mathbf{x_k})$ is deliberately left open: $\mathbf{g_k}$ could be randomly sampled from the entire set, specifically selected by a heuristic, or, to save computational time, just be the first subdifferential to be calculated. As such, $\mathbf{g_k}$ could be any subdifferential and all analysis will make no further assumptions.

When we compare Stochastic Gradient Descent and Stochastic Subgradient Descent, we can immediately see that the only distinction is that the former uses the gradient while the latter uses the subdifferential. In fact, when the function is smooth, the Properties of the Clarke Subdifferential shows that $\partial f(\mathbf{x}) = \{\nabla f(\mathbf{x})\}$ and so Stochastic Subgradient Descent simplifies to Stochastic Gradient Descent. Given these similarities, most general knowledge and practice transfers across immediately. For example, the simplicity of Stochastic Gradient Descent is maintained and so Stochastic Subgradient Descent acts as a good platform for further extensions to particular problem domains: already there exists extensions that employ linesearches and accelerated techniques [105, 107]. Furthermore, implementation and application of Stochastic Subgradient Descent is almost identical to Stochastic Gradient Descent, so many existing systems and frameworks can apply the algorithm with little or no significant change.

However, despite the similarities, there are still some difficulties. Most significantly, the change from the gradient to the Clarke subdifferential renders most theoretical analysis of Stochastic Gradient Descent inapplicable to Stochastic Subgradient Descent. In particular, the Convergence of Stochastic Gradient Descent does not apply as the preconditions for f are not met. As such, to avoid many of the same issues outlined in Applying Smooth Optimisation Techniques to Machine Learning, new theory and results must be developed.

3.2 Convergence

Given the importance of convergence results and the inability of Convergence of Stochastic Gradient Descent to immediately generalise to the subdifferential and non-smooth setting, we now dedicate time to presenting a convergence result for Stochastic Subgradient Descent. We develop this result in its entirety, stepping through the various issues and requirements that must be overcome to achieve the final convergence theorem. The final theorem is practical, but is rather abstract and so has limited immediate application. Therefore, we then outline various classes of functions that satisfy the theorem and so can be optimised using Stochastic Subgradient Descent.

The theory in this section follows the treatment given by Damek Davis et al. in 'Stochastic Subgradient Method Converges on Tame Functions' [36].

3.2.1 Proof Sketch

Before we leap into the proof of the convergence result, we first provide a sketch of the method employed.

The result is approached from the perspective of differential inclusions, i.e. set map generalisations of differential equations, and their discrete approximations. We model the iterates $\{\mathbf{x_k}\}_{k\geq 0}$ as approximations of an arc $z : \mathbb{R}_+ \to \mathbb{R}^d$ defined by the differential inclusion

$$z'(t) \in -\partial f(z(t)) \tag{3.2.1}$$

i.e. the arc is always moving downhill according to the Clarke Subdifferential. This model is illustrated in Figure 3.6 Given this model, we must check two requirements:

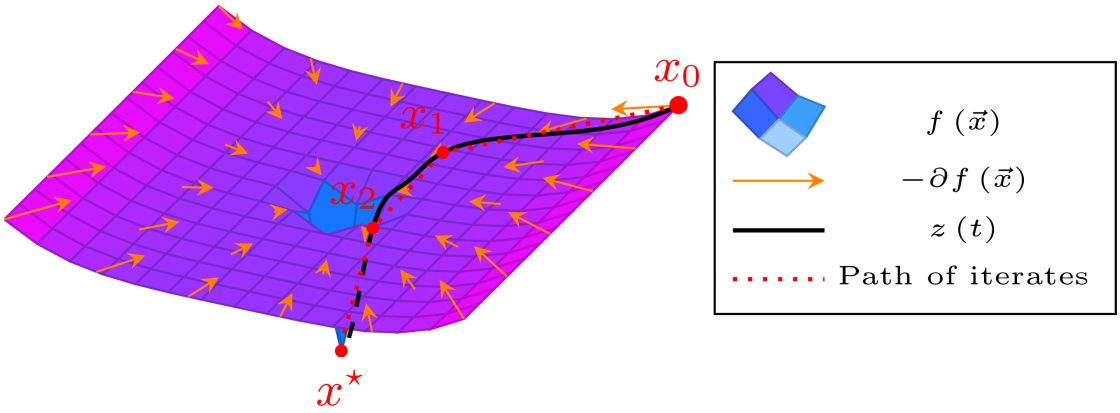


Figure 3.6: Modelling Iterates $\{\mathbf{x}_k\}_{k\geq 0}$ with Arc $z(t)$ defined by Downhill Flow according to Clarke Subdifferential

Convergence Requirements. The convergence of Stochastic Subgradient Descent depends upon the following two requirements being satisfied:

1. The iterates $\{\mathbf{x_i}\}_{0 \leq i \leq k}$ are discrete approximations of curves $x_k(\cdot)$ and the limit point of these curves $z(\cdot)$ is an arc satisfying the differential inclusion given in Equation (3.2.1).
2. The arc's limit point $z(\cdot)$ is a critical point of f: $0 \in -\partial f(z)$.

The first requirement simply ensures that the iterates will eventually trace out an arc that moves downhill according to the Clarke subdifferential. The second requirement ensures that the arc will reach a Clarke critical point, i.e. a local minimum, and then stabilise there.

Using the theory of differential inclusions, we develop two sets of assumptions that each enforces one of these requirements and thus ensure that a sequence of iterates will converge to a critical point. The first set of assumptions concerns the algorithm itself. The sequences of steps sizes $\{\alpha_k\}_{k \geq 0}$, the errors $\{\xi_k\}_{k \geq 0}$, and the iterates themselves $\{\mathbf{x_k}\}_{k \geq 0}$ must satisfy certain simple assumptions, such as boundedness and no significant bias, to ensure that the algorithm does eventually follow the limit point arc $z(\cdot)$. The second set of assumptions only concerns the objective function f, placing restrictions on what functions can be optimised using Stochastic Subgradient Descent. The primary restriction upon f is that it will eventually decrease if we follow the differentiation inclusion (3.2.1), i.e. move 'downhill' according to the Clarke Subdifferential.

Given these two sets of assumptions, we can ensure that a sequence of iterates will satisfy the convergence conditions. Therefore, we can put everything together to derive a practical convergence result for Stochastic Subgradient Descent.

3.2.2 Convergence of General Dynamic Systems

The proof method centres on differential inclusions and their discrete approximations. Thus, we begin by examining these constructions and then develop the theory for the assumptions that satisfy the Convergence Requirements. We undertake this task in a more general setting than Stochastic Subgradient Descent. This is done so that we can isolate the exact assumptions required for each convergence result and subsequently develop the most general and wide-reaching result for our setting.

Curves, Their Convergence, and Absolute Continuity

We begin with curves:

Definition 3.7 (Curve). *([36], Chp. 2, Sec. 1) Any continuous function $x : \mathbb{R}_+ \to \mathbb{R}^d$ is a curve in $\mathbb{R}^d$. The set $\mathcal{C}\left(\mathbb{R}_+, \mathbb{R}^d\right)$, i.e. the set of continuous functions from $\mathbb{R}_+$ to $\mathbb{R}^d$, is the set of curves of $\mathbb{R}^d$.*

Curves will be used as the continuous analogues of the sequence of iterates $\{\mathbf{x_k}\}_{k\geq 0}$, tracing out the path through the iterates on the graph of the objective function. Figure 3.7 illustrates this concept. With these curves constructed, we can reverse our perspective

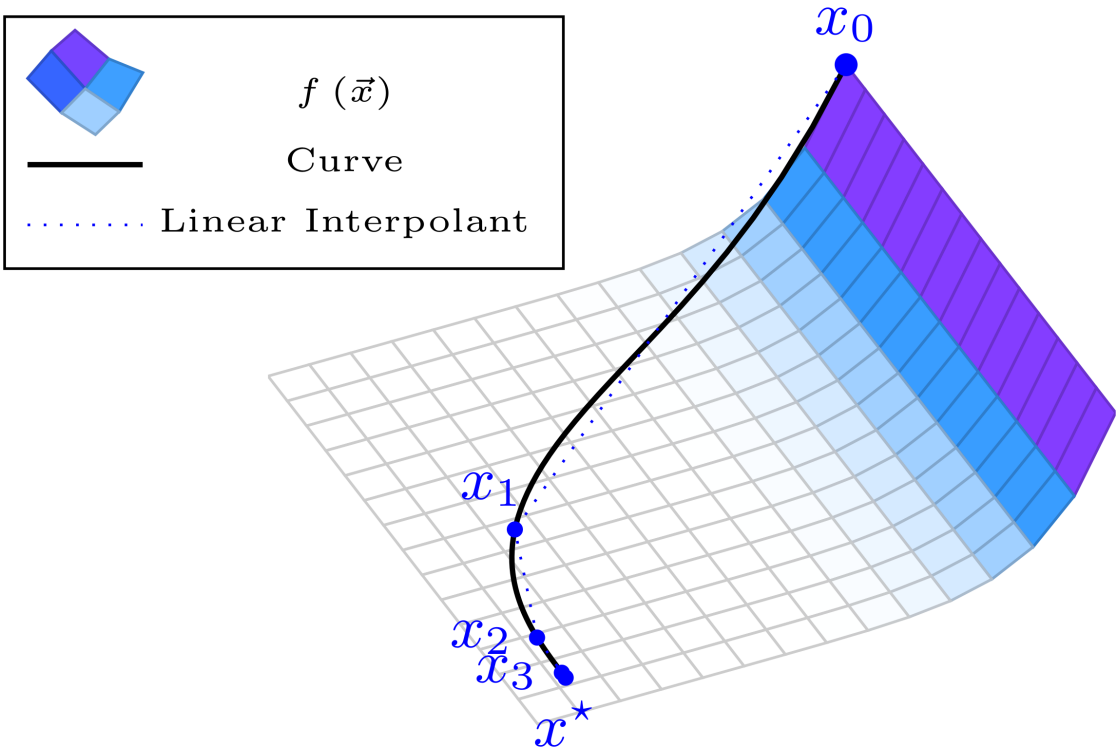


Figure 3.7: Smooth Curve and Linear Interpolation through Iterates

and view the curves as solutions to the differential inclusion (3.2.1) and the iterates as their discrete approximations. We will alternate between these two perspectives so that we can use both the machinery of discrete iterates and the machinery analytical curves to develop and prove the convergence results.

As any iterative algorithm runs, the set of iterates grows. As such, each iteration will produce a new curve $x_k : \mathbb{R}_+ \to \mathbb{R}^d$, tracing out a path through $\{\mathbf{x_i}\}_{0\leq i\leq k}$. Within the dual perspective, we need to determine when these curves converge so that their discrete approximations, i.e. the iterates, also converge. Thus, we must specify the assumptions of convergence for curves:

Definition 3.8 (Convergence of Curves)**.** *([36], Chp. 2, Sec. 1) A sequence of function* $\{f_k\}$ *converges to* f *in* $\mathcal{C}\left(\mathbb{R}_+, \mathbb{R}^d\right)$ *if* f_k *converges to* f *uniformly on compact intervals. That is, for all* $T > 0$*, we have that:*

$$\lim_{k\to\infty} \sup_{t\in[0,T]} \|f_k(t) - f(t)\|_2 = 0$$

At the moment, curves are not explicitly tied to any function and its flow over the space. Thus, we make the following definition to enforce such a link:

Definition 3.9 (Absolutely Continuity and Arcs)**.** *([36], Chp. 2, Sec. 1) A curve* $x : \mathbb{R}_+ \to \mathbb{R}^d$ *is* Absolutely Continuous *if there exists a map* $y : \mathbb{R}_+ \to \mathbb{R}^d$ *that is integrable on any compact interval and satisfies*

$$x(t) = x(0) + \int_0^t y(\tau)\, d\tau$$

for all $t \geq 0$*. Moreover, if this is the case, then* $x'(t) = y(t)$ *holds almost everywhere for* $t \geq 0$*. Such absolutely continuous curves are called* Arcs*.*

Absolute continuity is a stronger condition than standard continuity, but weaker than smooth, i.e. everywhere differentiable: $x'(t) = y(t)$ is only satisfied almost everywhere and so x' need only be defined almost everywhere. So any arc is continuous, being a curve, but isn't necessarily smooth. Note that if $f : \mathbb{R}^d \to \mathbb{R}$ is locally Lipschitz and x is an arc, then $f \circ x : \mathbb{R}_+ \to \mathbb{R}$ is an arc.

To eliminate unneeded complexity from our arguments, we use the most simple curve that traces out the path through the iterates:

Definition 3.10 (Linear Interpolation)**.** *Given a sequence of iterates* $\{\mathbf{x_k}\}_{k\geq 0}$ *and of step sizes* $\{\alpha_k\}_{k\geq 0}$*, we can construct the following curve:*

$$x(t) = \mathbf{x_k} + \frac{t - t_k}{t_{k+1} - t_k}\left(\mathbf{x_{k+1}} - \mathbf{x_k}\right) \qquad t \in [t_k, t_{k+1})$$

where $t_0 = 0$ *and* $t_m = \sum_{k=0}^{m-1} \alpha_k$*. We can also define the time-shifted curve* $x^\tau(x) = x(t + \tau)$ *for all* $\tau \geq 0$*.*

Note that although these linear interpolations are non-differentiable at countably many points, specfically every t_k for $k \geq 0$, they can still be arcs as they are still differentiable almost everywhere on its uncountable domain, the continuum $\mathbb{R}_+$. Figure (3.7) also illustrates a linear interpolation curve.

Functional Convergence of Discrete Approximations

With curves and arcs well defined, we can now turn our attention to verifying the assumptions that will enforce the first of the Convergence Requirements, i.e. the iterates produced by the algorithm will eventually follow an arc. To this end, we make the following definition:

Definition 3.11 (Trajectory)**.** *([36], Chp. 3, Sec. 1) Let $X \subseteq \mathbb{R}^d$ be a closed set and $G : X \rightrightarrows \mathbb{R}^d$ a set-valued map. Then an arc $x : \mathbb{R}_+ \to \mathbb{R}^d$ is a* Trajectory *of G if it satisfies the differential inclusion*

$$x'(t) \in G(x(t)) \tag{3.2.2}$$

almost everywhere for $t \geq 0$.

Note that this definition implies that the image of x is contained within X: if $\bar{t}$ is such that Equation (3.2.2) holds, then $x(\bar{t})$ must be in X for the equation to be well-defined; otherwise, we know that $\bar{t} \pm \varepsilon$ will satisfy the differential inclusion as $\varepsilon \to 0$ because the equation is satisfied almost everywhere, and so we merely need to apply the continuity of x and X being closed to infer that $x(\bar{t}) \in X$.

Given the definition of a trajectory, our goal is to determine what assumptions on the iterates allow a trajectory to be constructed through those iterates: or from the alternative perspective, what assumptions ensure that the iterates follow a trajectory. From either perspective, we are putting restrictions upon the algorithm producing the iterates. As our definition of a trajectory is general and not specific to any set-valued map G, we work with the following generalised iterative loop as our algorithm:

$$\mathbf{x_{k+1}} = \mathbf{x_k} + \alpha_k (\mathbf{y_k} + \xi_\mathbf{k}) \tag{3.2.3}$$

where $\{\mathbf{x_k}\}_{k\geq 0}$ is the sequence of iterates, $\{\alpha_k\}_{k\geq 0}$ is the sequence of step sizes, $\{\mathbf{y_k}\}_{k\geq 0}$ is the sequence of approximate evaluations of G at some point near $\mathbf{x_k}$, and $\{\xi_\mathbf{k}\}_{k\geq 0}$ is the sequence of errors. We note that this generalisation immediately applies to Stochastic Subgradient Descent by setting $G = -\partial f$, for some locally Lipschitz function f, and so the following results can be interpreted with this in mind.

Given our goal, we impose the following assumptions:

Assumption 3.1 (Generalised Iterate Assumptions)**.** *([36], Chp. 3, Sec. 1, Asm. A) We assume that the iterates $\{\mathbf{x_k}\}_{k\geq 0}$, approximate evaluations of G $\{\mathbf{y_k}\}_{k\geq 0}$, step sizes $\{\alpha_k\}_{k\geq 0}$, and errors $\{\xi_\mathbf{k}\}_{k\geq 0}$ of Equation (3.2.3) satisfy the following:*

1. *All limit points of $\{\mathbf{x_k}\}$ lie in X.*

   i.e. the algorithm will never converge outside of its domain.

2. *The iterates and approximations of G are bounded*

$$\sup_{k\geq 0} \|\mathbf{x_k}\|_2 < \infty \qquad \sup_{k\geq 0} \|\mathbf{y_k}\|_2 < \infty$$

i.e. the algorithm does not produce iterates or approximations of G that are infinite.

3. *The step sizes $\{\alpha_k\}_{k\geq 0}$ are non-negative, square summable, but not summable:*

$$\alpha_k \geq 0 \qquad \sum_{k\geq 0} \alpha_k = \infty \qquad \sum_{k\geq 0} {\alpha_k}^2 < \infty$$

i.e. the step sizes allow the algorithm to reach the entire domain, but they still converge to 0.

4. *The weighted noise sequence is convergent:*

$$\sum_{k\geq 0} \alpha_k \xi_{\mathbf{k}} \to \mathbf{v}$$

for some $\mathbf{v} \in \mathbb{R}^d$ as $k \to \infty$.

i.e. the noise sequence must not grow faster than the step sizes decrease.

5. *For any unbounded increasing sequence $\{k_j\} \subseteq \mathbb{N}$ such that $\mathbf{x_{k_j}}$ converges to some point $\bar{\mathbf{x}}$, it holds that:*

$$\lim_{n\to\infty} \operatorname{dist}\left(\frac{1}{n}\sum_{j=1}^{n} \mathbf{y_{k_j}}, G\left(\bar{\mathbf{x}}\right)\right) = 0$$

i.e. any convexification of the approximations $\{\mathbf{y_k}\}_{k\geq 0}$ converge to true value at the limit, $G\left(\bar{\mathbf{x}}\right)$.

The first two assumptions are obviously required, ensuring that the iterative loop does not produce unintelligible results. These assumptions are also very similar to the assumptions needed for Convergence of Stochastic Gradient Descent. The third assumption is the same as Step Size Sequence Convergence, ensuring that the algorithm does not converge before it can find its fixed point. The immediacy of the fourth and fifth assumptions is not as clear, but they prevent any noise or poor approximations from throwing the iterates off track from the differential inclusion (3.2.1). Additionally, these assumptions are not unreasonably strong and unexpected, echoing the restrictions of First and Second Moment Limits of Errors.

With these assumptions, we get the following result:

Theorem 3.5 (Functional Approximation)**.** *([36], Chp. 3, Sec. 1, Thm. 1) Suppose that the Generalised Iterate Assumptions hold when using the iterative loop given by Equation (3.2.3). Let $x(\cdot)$ be the curve given by the linear interpolation of the iterates, and $x^{\tau} = x(\tau + t)$ be the time-shifted curves for $\tau \geq 0$.*

Then, for any sequence $\{\tau_k\}_{k\geq 0} \subseteq \mathbb{R}_+$, the set of functions $\{x(\tau_k + \cdot)\}_{k\geq 0}$ is relatively compact, i.e. its closure is compact, in $\mathcal{C}(\mathbb{R}_+, \mathbb{R}^d)$. If in addition, $\tau_k \to \infty$ as $k \to \infty$, then all limit points $z(\cdot)$ of $\{x(\tau_k + \cdot)\}_{k\geq 0}$ in $\mathcal{C}(\mathbb{R}_+, \mathbb{R}^d)$ are trajectories of the differential inclusion (3.2.1).

We refer to Theorem 2 in 'Stochastic Methods for Composite Optimisation Problems' by John Duchi and Feng Ruan for a proof of the result [39].

This result is quite technical, but we can unpack it. The limit points $z(\cdot)$ can be intuitively understood to be the 'tails' of the linear interpolation $x(\cdot)$, produced by limiting the time-shifted curves by arbitrarily large τ's. These 'tails' are the only part of the curve that will converge as the step sizes for such high τ are arbitrarily small and so the iterates can model an arc. Note that despite these limit points being 'tails', they are still trajectories and so are well-defined for all $t \geq 0$: the time-shifted curves only cut off finite portions of the beginning and so the remaining portion of the curve extends infinitely with respect to time t. Finally, these limit points are formally known as subsequential limits, due to the method of their construction.

Given this unpacking, we can interpret the theorem as the following very simple and comprehensible statement: if the Generalised Iterate Assumptions hold, then the 'tail' of the linear interpolation $x(\cdot)$ is a trajectory. As such, the Functional Approximation Theorem sets out the exact assumptions required for an algorithm to produce iterates that follow a trajectory and so enforce the first of the Convergence Requirements in the general setting.

Subsequential Convergence to Critical Points

We now turn our attention to the second of the Convergence Requirements: the subsequential limits must be critical points. In the general setting, this requires that:

$$0 \in G(z)$$

where $z(\cdot)$ is a subsequential limit point. In these cases, the limit points set are constant trajectories at equilibrium.

The Functional Approximation Theorem allows for the existence of many subsequential limit points $z(\cdot)$ over the sequences $\{x(\tau_k + \cdot)\}_{k\geq 0}$ for $\{\tau_k\}_{k\geq 0} \subseteq \mathbb{R}_+$. Thus, we must guarantee that all possible limit points are critical points. Fortunately, such issues are common within continuous dynamics and so we can use their well-developed tool of

Lyapunov-like functions[3] to solve such a problem:

Assumption 3.2 (Generalised Lyapunov Assumptions). *([36], Chp. 3, Sec. 2, Asm. B) We assume that there exists a continuous function $\phi : \mathbb{R}^d \to \mathbb{R}$ which is bounded from below and such that the following two properties hold:*

1. *For a dense set of values $r \in \mathbb{R}$, the intersection $\phi^{-1}(r) \cap G^{-1}(0)$ is empty.*

   i.e. there exists a dense set of values in $\mathbb{R}$ that contains no critical values of G[a].

2. *Whenever $z : \mathbb{R}_+ \to \mathbb{R}^d$ is a trajectory of the differential inclusion (3.2.2) and $0 \notin G(x(0))$, there exists a real $T > 0$ satisfying*

$$\phi(z(T)) < \sup_{t \in [0,T]} \phi(z(t)) \leq \phi(z(0))$$

   i.e. the function ϕ eventually strictly decreases along the trajectories beginning at a non-critical point.

[a] Recall the distinction between critical points and critical values: Clarke Critical Points and Values

Using this tool, we derive the following result:

Theorem 3.6 (Convergence to Equilibrium Points). *([36], Chp. 3, Sec. 2, Thm. 2) Suppose that the Generalised Iterate Assumptions and Generalised Lyapunov Assumptions hold using the iterative loop given by Equation (3.2.3). Then every limit point of the iterates $\{\mathbf{x_k}\}_{k \geq 0}$ lies in $G^{-1}(0)$ and the function values $\{\phi(\mathbf{x_k})\}_{k \geq 0}$ converge.*

We refer to Section 3.3 in 'Stochastic Subgradient Method Converges on Tame Functions' by Damek Davis et al. for a proof of the result [36].

This theorem is far more digestible than the previous one, but there are still a few important aspects to note. Firstly, this result depends upon both sets of assumptions. The first set ensures that the iterates $\{\mathbf{x_k}\}_{k \geq 0}$ have subsequential limit points $z(\cdot)$, while the second ensures that these limit points converge to a critical point: together they ensure that the iterates themselves converge to a critical point. Additionally, the theorem also guarantees that the values of the Lyapunov-like function ϕ converge. Currently, this is insignificant - the Lyapunov-like function is merely a proof mechanic - however, we will use this later to ensure that the values of the objective function converge as well.

Ultimately, the Convergence to Equilibrium Points Theorem sets out the exact assumptions required for an algorithm to produce iterates whose limit points are critical points

[3] Aleksandr Mikhailovich Lyapunov (1857 - 1918) was a Russian mathematician who pioneered the development of stability theory of dynamical systems. Lyapunov functions are scalar functions used in a particular method for proving the stability of an ordinary differential equation.

and so enforce both of the Convergence Requirements, completing our goal in the general setting.

3.2.3 Convergence of Subdifferential Dynamic Systems

Generalised Iterate Assumptions and Generalised Lyapunov Assumptions provide the assumptions necessary for Theorems Functional Approximation and Convergence to Equilibrium Points. Collectively, these assumptions and results provide us with a systematic method for satisfying the Convergence Requirements and thus prove that the generated sequence of iterates will converge to a critical point. This method is general, enforcing minimal assumptions upon G or the various sequences. Consequently, we can use this method to prove that Stochastic Subgradient Descent converges.

We begin by highlighting the aspects of the general setting that we must specify for Stochastic Subgradient Descent. The first is that we must set $G = -\partial f$ so that the limit points $z(\cdot)$ satisfy:

$$z'(t) \in -\partial f(z(t))$$

As previously noted, this will ensure that the iterates are moving downhill according to the Clarke subdifferential. Secondly, the general iterative loop is refined to:

$$\mathbf{x_{k+1}} = \mathbf{x_k} - \alpha_k (\mathbf{y_k} + \xi_\mathbf{k})$$

where $\{\mathbf{x_k}\}_{k\geq 0}$ is the sequence of iterates, $\{\alpha_k\}_{k\geq 0}$ is the sequence of step sizes, $\{\mathbf{y_k}\}_{k\geq 0}$ is the sequence of Clarke subdifferentials of the iterates, i.e. $\mathbf{y_k} \in \partial f(\mathbf{x_k})$, and $\{\xi_\mathbf{k}\}_{k\geq 0}$ is the sequence of random variables that model the errors in sampling subdifferentials. These two aspects - setting G and refining the iterative loop - are the only specifications needed to apply the general setting to Stochastic Subgradient Descent.

We could stop here, simply substituting these specifications into our existing assumptions and theorems. However, the general results are not very practical: for example, assumption 4 of Generalised Iterate Assumptions can not be properly verified as it would require computing an infinite sum of values that are only known at runtime, and the implication of Convergence to Equilibrium Points only confirms the convergence of the Lyapunov-like function values, rather than f's values. Therefore, the current results are only helpful in a theoretical mathematical setting. Thus, we now develop a set of practical results that can be verified in reasonable assumptions and applied with little effort.

We begin by outlining a refined set of assumptions for the iterates:

Assumption 3.3 (Iterate Assumptions for Stochastic Subgradient Descent)**.** *([36], Chp. 4, Asm. C) We assume that the iterates $\{\mathbf{x_k}\}_{k\geq 0}$, subdifferentials $\{\mathbf{y_k}\}_{k\geq 0}$, step sizes $\{\alpha_k\}_{k\geq 0}$, and errors $\{\xi_\mathbf{k}\}_{k\geq 0}$ of Stochastic Subgradient Descent satisfy the following:*

1. *The sequence* $\{\alpha_k\}_{k\geq 0}$ *is non-negative, square summable, but not summable:*

$$\alpha_k \geq 0 \qquad \sum_{k\geq 0} \alpha_k = \infty \qquad \sum_{k\geq 0} {\alpha_k}^2 < \infty$$

i.e. the step sizes allow the algorithm to reach the entire domain, but they still converge to 0*.*

2. *Almost surely*[a]*, the iterates are bounded:*

$$\sup_{k\geq 0} \|\mathbf{x_k}\|_2 < \infty$$

i.e. the algorithm produces finite iterates 100% *of the time.*

3. *The error sequence* $\{\xi_\mathbf{k}\}_{k\geq 0}$ *is a martingale difference sequence with respect to the increasing* σ*-fields*

$$\mathcal{F}_k = \sigma\left(\mathbf{x_j}, \mathbf{y_j}, \xi_\mathbf{j} : j \leq k\right)$$

That is, there exists a function $p : \mathbb{R}^d \to \mathbb{R}_+$ *which is bounded on bounded sets, such that, almost surely, for all* $k \in \mathbb{N}$*, we have:*

$$\mathbb{E}\left[\xi_\mathbf{k} \mid \mathcal{F}_k\right] = 0 \qquad \mathbb{E}\left[\|\xi_\mathbf{k}\|_2 \mid \mathcal{F}_k\right] \leq p\left(\mathbf{x_k}\right)$$

i.e. the noise random variables are not dependent upon the past and are gently bounded.

[a] Another technical term from measure theory that roughly means: the set of settings where the event does happen is so large in comparison to the set of settings where the event does not happen that there is 100% probability of the event occurring, e.g. you will almost surely choose an irrational number when randomly sampling from the number line $\mathbb{R}$.

In comparison to the Generalised Iterate Assumptions, this set of assumptions is much more practical. Most notably, assumption 3 is a property of random variables that can be verified relatively easily from the approximation sampling process, rather than through an infinite runtime sum. However, we do need to verify that this practical set of assumptions satisfies the general assumptions:

Lemma 3.1 (Almost Sure Implication of Iterate Assumptions)**.** *([36] Chp. 4, Lmm. 1) The Iterate Assumptions for Stochastic Subgradient Descent guarantee that the Generalised Iterate Assumptions holds almost surely.*

We refer to Lemma 4.1 in 'Stochastic Subgradient Method Converges on Tame Functions' by Damek Davis et al. for a proof of the result [36].

We note that this lemma does have the 'almost surely' qualifier. Technically, this allows for problem instances that will satisfy the practical assumptions but will fail the general

assumptions. However, while these instances can exist, the chance of them being realised is 0% and so they are not a practical concern. Thus, Iterate Assumptions for Stochastic Subgradient Descent outlines the practical assumptions needed to satisfy the first of Convergence Requirements for Stochastic Subgradient Descent.

We now endeavour to develop a practical analogue for Generalised Lyapunov Assumptions. Fortunately, given our existing restrictions upon f, we can use it as our Lyapunov-like function:

Assumption 3.4 (Lyapunov Assumptions for Stochastic Subgradient Descent)**.** *([36], Chp. 4, Asm. D) We assume that the objective function f used in the Stochastic Subgradient Descent satisfies the following two properties:*

1. *The set of non-critical values of f is dense in $\mathbb{R}$.*
2. *Whenever $z : \mathbb{R}_+ \to \mathbb{R}^d$ is a trajectory of the differential inclusion $z' \in -\partial f(z)$ and $z(0)$ is itself not a critical point of f, there exists a real $T > 0$ satisfying*

$$f(z(T)) < \sup_{t \in [0,T]} f(z(t)) \leq f(z(0))$$

i.e. the objective function f eventually strictly decreases along trajectories beginning at non-critical points.

This practical set of assumptions is very similar to its counterpart Generalised Lyapunov Assumptions: substituting f for the Lyapunov-like function makes the specification relatively simple. The one additional change needed is for the first of the two assumptions. Rather than requiring that there exists a dense set of values in $\mathbb{R}$ that do not contain a critical value, we require the slightly stronger property that the set of non-critical values is dense. This formulation is much more practical - existence properties can be very difficult to verify without a constructionist method - but it also is not very strong overall: a standard and general result in analysis, Sard's theorem [87], states the far stronger property that the set of critical values of a sufficiently smooth function has measure 0.

For sake of completion, we present:

Lemma 3.2 (Implication of Lyapunov Assumptions)**.** *The Lyapunov Assumptions for Stochastic Subgradient Descent guarantee that the Generalised Lyapunov Assumptions holds.*

Its proof follows by the arguments within our previous discussion. However, this lemma does confirm that the practical Lyapunov Assumptions for Stochastic Subgradient Descent will enforce the Convergence Requirements for Stochastic Subgradient Descent.

Ultimately, we have derived two new sets of assumptions, Iterate Assumptions for Stochastic Subgradient Descent and Lyapunov Assumptions for Stochastic Subgradient

Descent, which imply the general assumptions need to guarantee convergence. Thus, we can prove:

Theorem 3.7 (Convergence of Stochastic Subgradient Descent Under Practical Assumptions)**.** *([36], Chp. 4, Thm. 2) Suppose that the Iterate Assumptions for Stochastic Subgradient Descent and Lyapunov Assumptions for Stochastic Subgradient Descent hold for an application of Stochastic Subgradient Descent. Then, almost surely, every limit point of the iterates* $\{\mathbf{x_k}\}_{k\geq 0}$ *is critical for* f *and the function values* $\{f(x_k)\}_{k\geq 0}$ *converge.*

We refer to Theorem 4.2 in 'Stochastic Subgradient Method Converges on Tame Functions' by Damek Davis et al. for a proof of the result [36].

This theorem provides us with a convergence result for Stochastic Subgradient Descent. Furthermore, this result is relatively practical: once the concrete set of assumptions has been checked, we can use the algorithm with the certainty that it will indeed converge to a local minimum. Thus, we have succeeded in presenting the important convergence result for the practical use of Stochastic Subgradient Descent.

3.2.4 Sufficient Classes of Functions

Convergence of Stochastic Subgradient Descent Under Practical Assumptions gives a practical convergence result for Stochastic Subgradient Descent. However, some aspects of the result are still relatively abstract: while the Iterate Assumptions for Stochastic Subgradient Descent are concrete and can be determined from the algorithm's exact setup and instance, the Lyapunov Assumptions for Stochastic Subgradient Descent merely set out mathematical properties that f must satisfy. Fortunately, the first of the Lyapunov Assumptions for Stochastic Subgradient Descent is simple and easy to verify computationally: unfortunately, the second - the objective function eventually strictly decreases along trajectories beginning at non-critical points - can only be verified through mathematical proof. It would be far better to have an explicit class of functions for which the properties hold, bypassing the need to conduct a mathematical proof to guarantee a function's convergence. Functions in this class could then be optimised using Stochastic Subgradient Descent with ease.

We do this by first identifying a different mathematical property that implies the second assumption of the Lyapunov Assumptions for Stochastic Subgradient Descent: the Trajectory. This is a stronger but far more tangible and established property, so we can use it to outline and prove two classes of functions that satisfy the assumptions: Regular Functions and Whitney Stratifiable Functions.

Chain Rule Satisfies Assumptions

We begin with the formal definition of the property in question:

Definition 3.12 (Chain Rule). *([36], Chp. 5, Def. 1) Let $f : \mathbb{R}^d \to \mathbb{R}$ be a locally Lipschitz function. We say that f admits a* Chain Rule *if for any arc $z : \mathbb{R}_+ \to \mathbb{R}^d$ the equality*

$$(f \circ z)'(t) = \langle \partial f(z(t)), z'(t) \rangle$$

holds almost everywhere for $t \geq 0$.

This definition should be familiar: it is a generalisation of the standard chain rule from analysis

$$h = f \circ g \implies h' = (f' \circ g) \cdot g'$$

where $\cdot$ denotes the dot product. Intuitively, such a property should infer the second assumption of the Lyapunov Assumptions for Stochastic Subgradient Descent: if the function admits the chain rule, then the Clarke subdifferential and arc separate into distinct terms within the inner-product and so the result should go downhill along the trajectory. Fortunately, we can prove this result:

Lemma 3.3 (Chain Rule Implies Descent). *([36], Chp. 5, Lmm. 2) Let $f : \mathbb{R}^d \to \mathbb{R}$ be a locally Lipschitz function that admits a chain rule. Let $z : \mathbb{R}_+ \to \mathbb{R}^d$ be any arc satisfying the differential inclusion*

$$z'(t) \in -\partial f(z(t))$$

almost everywhere for $t \geq 0$. Then the equality $\|z'(t)\|_2 = \operatorname{dist}(0, \partial f(z(t)))$ holds almost everywhere for $t \geq 0$ and therefore

$$f(z(0)) - f(z(t)) = \int_0^t \left(\operatorname{dist}(0, \partial f(z(\tau))) \right)^2 d\tau$$

holds for all $t \geq 0$. In particular, the second assumption of the Lyapunov Assumptions for Stochastic Subgradient Descent holds.

We refer to Lemma 5.2 in 'Stochastic Subgradient Method Converges on Tame Functions' by Damek Davis et al. for a proof of the result [36].

This lemma does enable us to use the chain rule to outline a satisfactory class of functions. However, the lemma also shows that the chain rule is a stronger property than the second assumption of the Lyapunov Assumptions for Stochastic Subgradient Descent. Therefore, there may be functions that we are eliminating from consideration by using this stronger property. Consequently, it is worth motivating why we use the chain rule as a stepping stone. The principal reason is ease: it is a well-known property and so there exists machinery for verifying that a function satisfies its conditions. This is not true for the second assumption of the Lyapunov Assumptions for Stochastic Subgradient Descent and so outlining a large class of satisfactory functions would be difficult beyond the utility gained. Furthermore, we can always default to the Convergence of Stochastic

Subgradient Descent Under Practical Assumptions whenever there is a function that does not satisfy the chain rule yet needs to be optimised. Thus, there is no loss in using the chain rule as a stepping stone in our proofs and we only gain ease and limit complexity.

We now use the Chain Rule Implies Descent to outline two classes of functions that will satisfy the second assumption of the Lyapunov Assumptions for Stochastic Subgradient Descent. The first is the class of Regular Functions and the second is the class of Whitney Stratifiable Functions.

Chain Rule under Regular Functions

It should be no surprise that Regular Functions admit a chain rule and so satisfy the second assumption of Lyapunov Assumptions for Stochastic Subgradient Descent: by definition, their Clarke directional derivative agrees with the standard directional derivative in all directions at all points, so their Clarke subdifferential is merely a convexification of the standard gradient. However, despite such immediate intuition, we still prove the result:

Lemma 3.4 (Chain Rule under Regular Functions)**.** *([36], Chp. 5, Sec. 1, Lmm. 4) Any locally Lipschitz function that is regular admits a chain rule and therefore satisfies the Descent property of Lyapunov Assumptions for Stochastic Subgradient Descent.*

We refer to Lemma 5.4 in 'Stochastic Subgradient Method Converges on Tame Functions' by Damek Davis et al. for a proof of the result [36]. Notably, in addition to proving our intuition, this result and its proof sheds further light on the relation between the Clarke subdifferential and the standard gradient: in the convex case, a Clarke subdifferential can be viewed as a sub-first order[4] approximation of the gradient.

With this lemma, we immediately can prove the following:

Corollary 3.1 (Convergence of Stochastic Subgradient Descent on Regular Functions)**.** *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a locally Lipschitz function that is also regular. Suppose that f's noncritical values are dense in $\mathbb{R}$. Suppose that the iterates $\{\mathbf{x_k}\}_{k\geq 0}$ produced by Stochastic Subgradient Descent satisfy the Iterate Assumptions for Stochastic Subgradient Descent. Then almost surely, every limit point of the iterates $\{\mathbf{x_k}\}_{k\geq 0}$ is critical for f and the function values $\{f(\mathbf{x_k})\}_{k\geq 0}$ converge.*

This is a significant result: Regular Functions is a large class of functions, encompassing convex functions and more. However, any function with a locally non-convex region is not in the class. This eliminates many pseudoconvex or even quasiconvex functions,

[4] Sub-first order in the sense that it produces an approximation that is always less than the true value. This directly extends our intuition from the Lipschitz and Convexity Imply Regularity result.

the former of which ought to be simple to optimise by definition. For example, the pseudoconvex function

$$f(x, y) = (|x| - |y|)^2$$

is locally concave along $x = 0$ and $y = 0$ and so convergence is not guaranteed by Convergence of Stochastic Subgradient Descent on Regular Functions. Figure 3.8 illustrates

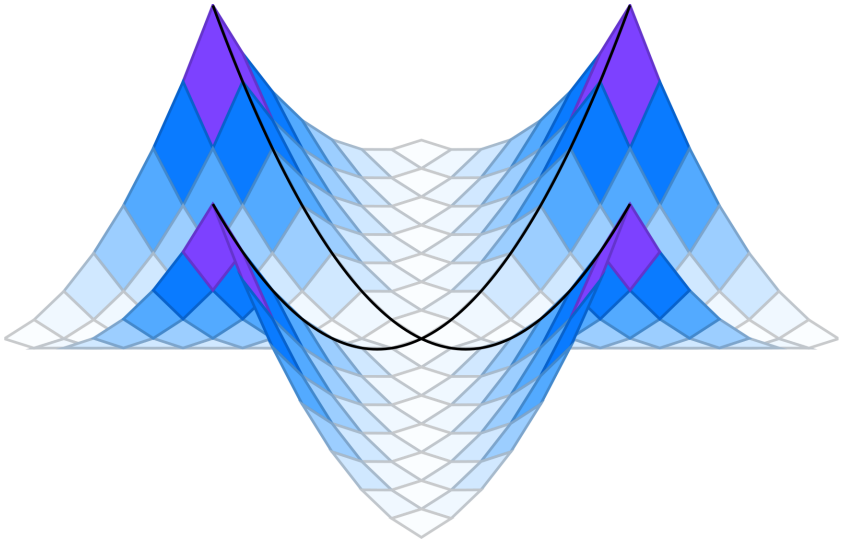


Figure 3.8: Non-regular function $f(x, y) = (|x| - |y|)^2$: Regions along the ridges defined by the black lines are non-convex.

this function and highlights its non-convex regions. Therefore, while Convergence of Stochastic Subgradient Descent on Regular Functions is a strong and useful result, we move on to find a wider class of functions and a stronger corollary.

Stratifiable Functions

The next class of functions is significantly wider, focusing on a global property rather than a local property like that of Regular Functions. However, this does come at a cost: the mathematical definition of this class is complex, being based in the field of differential geometry. This forces us to develop a few complicated ideas to define the class. As our purpose is to connect the realm of machine learning and mathematical optimisation, we will not focus on these technical definitions, but rather identify the key implications for practical applications of Stochastic Subgradient Descent. A complete mathematical consideration of these ideas can be found in 'Clarke Subgradient of Stratifiable Functions' by J. Bolte et Al [17].

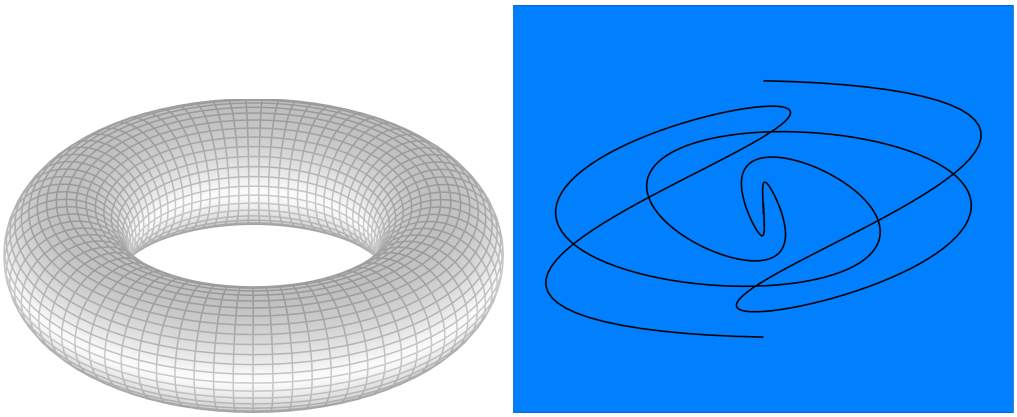
We begin this adventure with a cursory introduction to differential manifolds:

Definition 3.13 (Smooth Manifolds)**.** *([36], Chp. 5, Sec. 2) A set $M \subseteq \mathbb{R}^d$ is a C^p* Smooth Manifold *if there is an integer $n \in \mathbb{N}$ such that around any point $\mathbf{x} \in M$, there is a neighbourhood $U_{\mathbf{x}} \subseteq \mathbb{R}^d$ and a C^p-smooth, i.e. p-times differentiable, map $F : U \to \mathbb{R}^{d-n}$ with $\nabla F(\mathbf{x})$ of full rank and satisfying $M \cap U = \{y \in U : F(y) = 0\}$. In this case, the* Tangent *and* Normal Spaces *to M at $\mathbf{x}$ are defined to be*

$$T_M(\mathbf{x}) = \text{Null}(\nabla F(\mathbf{x}))$$
$$N_M(\mathbf{x}) = (T_M(\mathbf{x}))^{\perp}$$

i.e. the tangent space is the set of values in U such that the gradient of F maps to 0, and the normal space is the set of values orthogonal to the tangent space.

Manifolds can intuitively be understood as surfaces with some ambient space: the surface of a doughnut is a 2 dimensional manifold within $\mathbb{R}^3$, the volume of the same doughnut is a 3 dimensional manifold within $\mathbb{R}^4$, and a curve on paper is a one dimensional manifold within $\mathbb{R}^2$. Figure Example Manifolds illustrates these examples. As the second example



(a) Doughnut Manifold (b) Curve Manifold

Figure 3.9: Example Manifolds

shows, once we move beyond 3 dimensional spaces, intuition can be difficult. However, intuition from lower dimensional spaces generally moves to higher dimensional spaces and so we encourage a reader to focus on these lower dimensional examples.

A smooth manifold is a manifold for which we can define a calculus upon the surface. This can only be done if the manifold is smooth in the natural sense, i.e. does not have local areas that are discontinuous or change direction instantly. The calculus is given by the maps F and their gradients ∇F seen in the formal definition. Given a smooth manifold, the tangent space $T_M(\mathbf{x})$ is the set of gradients that represent a direction along the manifold an arbitrarily small amount, while the normal space $N_M(\mathbf{x})$ is the set of gradients that represent a direction immediately above or below the manifold an

arbitrarily small amount. Any gradient can be expressed as the weighted sum of a vector from the tangent space and a vector from the normal space.

We use smooth manifolds to make following definition:

Definition 3.14 (Whitney Stratification)**.** *([36], Chp. 5, Sec. 2, Def. 6) A* Whitney C^p-Straification $\mathcal{A}$ *of a set* $Q \subseteq \mathbb{R}^d$ *is a partition of* Q *into finitely many non-empty* C^p *manifolds, called* Strata*, satisfying the following conditions:*

1. Frontier Condition*: All pairs of strata* L *and* M *satisfy:*

$$L \cap \mathrm{cl}\,(M) \neq \varnothing \implies L \subseteq \mathrm{cl}\,(M)$$

   i.e. if one stratum L *intersects the closure of another* M*, then* L *must be fully contained in the closure of* M*. This endows the strata with a partial order:* $L \succeq M \iff L \subseteq \mathrm{cl}\,(M)$*.*

2. Whitney Condition*: For any sequence of points* $\mathbf{z_k}$ *in a stratum* M *converging to a point* $\mathbf{z}$ *in a stratum* L*, if the corresponding normal vectors* $\mathbf{n_k} \in N_m(\mathbf{z_k})$ *converge to a vector* $\mathbf{v}$*, then* $\mathbf{v} \in N_L(\mathbf{z})$ *holds*

   i.e. the limit of normals along a sequence of manifold points in a stratum must be a normal to the stratum containing the limit of the manifold points.

Intuitively, a Whitney C^p-stratification of a smooth manifold is a sequence of successively large subsets of the manifold upon which limits between the subsets behave well. For the most part, a smooth manifold does not admit a Whitney stratification only if it has pathological behaviour, either in its surface or its calculus.

Fortunately, our next definition is far simpler:

Definition 3.15 (Whitney Stratifiable Functions)**.** *A function* $f : \mathbb{R}^d \to \mathbb{R}$ *admits a Whitney* C^p*-stratification if its graph* $G_f = \left\{(\mathbf{x}, y) \in \mathbb{R}^d \times \mathbb{R} : f(\mathbf{x}) = y\right\}$ *admits a Whitney* C^p*-stratification.*

We can use our intuition for Whitney stratifications to develop a rough intuition for Whitney stratifiable functions. We highlight that the class of Whitney stratifications is very wide and encompassing, as only pathological manifolds do not admit any Whitney stratification. In a moment, we will outline how wide this class is using explicit examples, but currently, we develop the required lemmas to apply this class to Convergence of Stochastic Subgradient Descent Under Practical Assumptions:

Lemma 3.5 (Chain Rule under Statifications). *([36], Chp. 5, Sec. 2, Thm. 8) Any locally Lipschitz function $f : \mathbb{R}^d \to \mathbb{R}$ that is Whitney C^1-stratifiable admits a chain rule, and therefore satisfies the second assumption of the Lyapunov Assumptions for Stochastic Subgradient Descent.*

We refer to Theorem 5.8 in 'Stochastic Subgradient Method Converges on Tame Functions' by Damek Davis et al. for a proof of the result [36].

Already, this lemma allows us to develop a corollary similar to Convergence of Stochastic Subgradient Descent on Regular Functions. However, we first identify another helpful result:

Lemma 3.6 (Critical Values under Stratifications). *([36], Chp. 5, Sec. 2, Lmm. 7) The set of critical value of any locally Lipschitz Whitney C^d-stratifiable function $f : \mathbb{R}^d \to \mathbb{R}$ has zero measure. In particular, the first assumption of the Lyapunov Assumptions for Stochastic Subgradient Descent holds.*

We refer to Lemma 5.7 in 'Stochastic Subgradient Method Converges on Tame Functions' by Damek Davis et al. for a proof of the result [36].

Given these two lemmas, a Whitney C^d-stratifiable function immediately satisfies all of the Lyapunov Assumptions for Stochastic Subgradient Descent. Technically, the second lemma imposes stronger conditions upon f than the former: a Whitney C^p-stratifiable function is always C^1-stratifiable, but the converse does not hold. However, the stronger condition is minimal and the benefits are substantial: all assumptions upon the objective functions are eliminated, leaving only the algorithmic ones within the Iterate Assumptions for Stochastic Subgradient Descent. Thus, we get the following powerful result:

Corollary 3.2 (Convergence of Stochastic Subgradient Descent on Tame Functions). *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a locally Lipschitz function that is also Whitney C^d-stratifiable. Suppose that the iterates $\{\mathbf{x_k}\}_{k\geq 0}$ produced by Stochastic Subgradient Descent satisfy the Iterate Assumptions for Stochastic Subgradient Descent. Then almost surely, every limit point of the iterates $\{\mathbf{x_k}\}_{k\geq 0}$ is critical for f and the function values $\{f(\mathbf{x_k})\}_{k\geq 0}$ converge.*

This is a phenomenal result that has wide implications for the application of Stochastic Subgradient Descent. However, these implications are currently shrouded by the mathematical complexity of Whitney stratifications. So we now outline the class of Whitney stratifiable functions and highlight some key examples.

An important subclass is the following:

Definition 3.16 (Semialgebraic Function)**.** *A function* $f : \mathbb{R}^d \to \mathbb{R}$ *is* Semialgebraic *if its graph* G_f *can be defined by the finite set of polynomial equations and inequalities. That is:*

$$G_f = \left\{ (\mathbf{x}, y) \in \mathbb{R}^d \times \mathbb{R} : \forall i \in I, P_i(\mathbf{x}, y) = 0, \forall j \in J, Q_j(\mathbf{x}, y) > 0 \right\}$$

where I *and* J *are finite index sets and* $P_i, Q_j : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}$ *are* $d+1$ *dimensional polynomials.*

Although this is another technical definition, the class is simple to identify: semialgebraic functions are polynomials and functions defined piecewise with finitely many polynomials. Already, this is very wide class of functions covering many objective functions found in practical applications, for example neural networks using ReLU as their activation function.

However, semialgebraic functions are not exhaustive and do exclude some key functions. For example, $f(x) = e^x$ is not semialgebraic, as the exponential is the sum of infinitely many polynomials:

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

Fortunately, Whitney stratifiable functions encompass an even wider class than the semialgebraic functions. This class is defined within the following model:

Definition 3.17 (o-Minimal Structure)**.** *([36] Chp. 5, Sec. 2, Def. 10) An* o-Minimal Structure *is a sequence of Boolean algebras* $\mathcal{O}_d$ *of subsets of* $\mathbb{R}^d$[a] *such that for each* $d \in \mathbb{N}$*:*

1. *If* $A \in \mathcal{O}_d$*, then* $A \times \mathbb{R}, \mathbb{R} \times A \in \mathcal{O}_{d+1}$*;*
2. *If* $\pi : \mathbb{R}^d \to \mathbb{R} \to \mathbb{R}$ *denotes the coordinate projection onto* $\mathbb{R}^d$*, i.e.* $\pi(\mathbf{x}, r) = \mathbf{x}$*, then for any* $A \in \mathcal{O}_{d+1}$*, we have* $\pi(A) \in \mathcal{O}_d$*;*
3. $\mathcal{O}_d$ *contains all sets of the form* $\left\{ \mathbf{x} \in \mathbb{R}^d : p(\mathbf{x}) = 0 \right\}$*, where* p *is a polynomial on* $\mathbb{R}^d$*; and,*
4. *The elements of* $\mathcal{O}_1$ *are exactly the finite unions of intervals (possibly infinite) and points.*

The sets A *belonging to* $\mathcal{O}_d$*, for some* $d \in \mathbb{N}$*, are called* Definable *in the o-minimal structure.*

[a] A Boolean algebra over a set X is a set $\mathcal{O}$ of subsets of X, i.e. $\mathcal{O} \subseteq 2^X$, such that $\mathcal{O}$ is closed under union, intersection and complement (with respect to X).

While this is a technical and relatively obtuse definition from model theory, we only need to concern ourselves with the following: Andrew Wilkie showed that there exists

an o-minimal structure that contains all semialgebraic functions and the exponential function [102]. Thus, o-minimal structures allow for a very disparate variety of functions to be defined. This class of functions is given a specific name:

Definition 3.18 (Tame Functions)**.** *A function $f : \mathbb{R}^d \to \mathbb{R}$ is* Tame *if the intersection of its graph G_f and any ball is definable within some o-minimal structure.*

A proof that all tame functions admit a Whitney stratification can be found in 'Geometric Categories and O-Minimal Structures' by L. van den Dries and C. Miller [97]. Therefore Convergence of Stochastic Subgradient Descent on Tame Functions applies to an incredibly wide class of functions and is consequently very significant for the application of Stochastic Subgradient Descent.

3.3 Discussion of the Theory

While developing the theory throughout this chapter, we have noted and highlighted various aspects of the definitions and results. However, each of these has been considered in isolation from one another. So to conclude this chapter, we briefly discuss the theory in its entirety, addressing some recurrent ideas and issues.

The issues that we address are as follows: the continuous presence of 'almost everywhere' and its implications, most notably in relation to Rademacher's Theorem; the limits of the Iterate Assumptions for Stochastic Subgradient Descent and Lyapunov Assumptions for Stochastic Subgradient Descent; the computability of the Clarke Subdifferential; sampling methods for the Clarke Subdifferential; and finally, the application of Stochastic Subgradient Descent to machine learning, in particular we derive a novel corollary that establishes the conditions for its application to General Supervised Learning methods.

We then conclude this section and chapter with a summary of the theory of Stochastic Subgradient Descent.

3.3.1 Almost Everywhere

Throughout the theory, 'almost everywhere' and 'almost surely' have appeared regularly. So far, we have moved past each instance without concern, but it is worth properly clarifying these technical measure theory terms and analysing each instance's implications.

'Object o is property P almost everywhere on space S' can roughly be interpreted as: if we randomly sample a point $s \in S$, then the chance of object o satisfying P at s is 100%. This does not preclude the existence of points $s' \in S$ where o does not satisfy P, but they are so few that the chance of their random appearance is negligible. On the other hand, 'object $o \in O$ is property P almost surely' can be roughly translated as: the chance of $o \in O$ satisfying P is 100%. Again, this does not preclude the existence of objects $o' \in O$ that do not satisfy P, but they are so few that the chance of their

random appearance is also negligible. The difference between the two terms is minor: 'almost everywhere' is used when the property depends upon an ambient space, while 'almost surely' is used when the property does not depend upon an ambient space and so the object varies by itself. Therefore, the two terms have very similar meanings.

Given these meanings, we can interpret and analyse the implications of these terms throughout the theory. The first and most significant appearance is in Rademacher's Theorem. In this theorem, the inclusion of 'almost everywhere' implies that locally Lipschitz functions are differentiable everywhere except on a very small set of points, the chance of randomly choosing which is 0%. Given this, we may question the purpose of Stochastic Subgradient Descent: why do we need an algorithm that applies to non-smooth locally Lipschitz functions, if the chance of randomly ending up on a non-smooth point is 0%? Can we not just use Stochastic Gradient Descent and be almost certain that all points will be differentiable? There are two responses to these questions.

Firstly, there are settings for which the non-differentiable points are unavoidable, despite their small number. For example, if the optimal point is non-differentiable, e.g. the absolute value function $f(x) = |x|$, then non-differentiable points are necessarily unavoidable and so we must consider the theory of non-smooth optimisation and apply Stochastic Subgradient Descent to guarantee convergence. There are other settings for which non-differentiable points are unavoidable: if the initial point is set by physical requirements, it may be forced to be non-differentiable; or, the iterates may naturally move towards non-differentiable points due to the topology of the function, e.g. using Stochastic Subgradient Descent on $f(\mathbf{x}) = \|x\|_1$ leads to many iterates being non-differentiable points. Fundamentally, a function being differentiable almost everywhere only implies that a randomly selected point will be differentiable with 100% certainty. However, descent methods do not randomly sample points and so the chance of an iterate being non-differentiable is non-zero.

Secondly, even if the mathematical chance of an iterate being non-differentiable when using Stochastic Gradient Descent is 0%, the practical chance when computing the algorithm may be non-zero due to rounding errors or other computational limits. Practical applications of descent algorithms do not use the real number line and perfect mathematical operations, but rather finite valued precision on a computer with a simple bit-adder. As a result, a differentiable iterate $\mathbf{x_k}'$ may be recorded as a non-differentiable point $\mathbf{x_k}$ due to some computational limit. Thus, even if mathematically the non-differentiable points should not appear when applying an algorithm, they may appear regardless. For a specific example, we can consider ReLU. Its only non-differentiable point is at $x = 0$ and so it is differentiable almost everywhere:

$$\mathrm{ReLU}'(x) = \begin{cases} -1 & x > 0 \\ 0 & x < 0 \end{cases}$$

Therefore, for a given sequence of step sizes $\{\alpha_k\}_{k\geq 0}$, there are only countably many initial points $\left\{\mathbf{x_0^k}\right\}_{k\geq 0}$ that will result in a future iterate being 0. As an initial point can

come from the entire uncountable number line $\mathbb{R}$, the mathematical chance of an iterate being non-differentiable is still 0 when using Stochastic Subgradient Descent. However, if the computer uses single precision values, then we only have at most 8 decimal digits of precision [1] and so any value in $\left[-1 \times 10^{-8}, 1 \times 10^{-8}\right]$ will round to 0. In turn, this means that any initial point within the intervals

$$\left\{ \left[\left(1 - 10^{-8}\right) \mathbf{x_0^k}, \left(1 + 10^{-8}\right) \mathbf{x_0^k} \right] \right\}_{k \geq 0}$$

may lead to an iterate being 0. Consequently, the practical chance of an iterate being non-differentiable when using Stochastic Subgradient Descent on ReLU may be non-zero. Hence, the mathematical chance of an iterate being non-differentiable does not represent the practical chance, and Stochastic Subgradient Descent may be required in settings where these chances do not align.

The paper 'Conservative Set Valued Fields, Automatic Differentiation, Stochastic Gradient Methods and Deep Learning' by J. Bolte and E. Pauwels [20] considers this second response in more depth. They complete an experimental test using neural networks with ReLU activations functions to determine that only 5 layers are needed before the chance of an iterate sitting at a non-differentiable point is over 50%.

Given these two responses, we can see that non-smooth optimisation theory and Stochastic Subgradient Descent are a necessity. Therefore, while Rademacher's Theorem does imply that all locally Lipschitz functions are differentiable almost everywhere, the result does not diminish the impact and significance of Stochastic Subgradient Descent.

We briefly note that the Clarke Subdifferential can be extended to functions that are merely bounded above, and therefore not necessarily differentiable almost everywhere or even continuous. This is done by treating the Clarke Subdifferential by Directional Derivatives Theorem as the primary definition: the bounded above condition exists as the lim sup requires the upper bound. Hence, we can apply Stochastic Subgradient Descent to functions that are not differentiable almost everywhere. However, there is no convergence theory for such an application as we have seen that the current results rely upon f being locally Lipschitz. Future research into the convergence of Stochastic Subgradient Descent in this extended setting would be useful.

We now return to our examination of 'almost everywhere' and 'almost surely': fortunately, the remaining instances are far less contentious. Both Absolutely Continuity and Arcs and Trajectory have 'almost everywhere' in their definitions which relate to the arc satisfying a differential inclusion. This is done to allow arcs that diverge from the differential inclusion at isolated points. These divergences do not impact the path of the arc and so the inclusion of 'almost everywhere' simply expands the set of arcs with no loss. The definition of Chain Rule and an associated Lemma Chain Rule Implies Descent include 'almost everywhere' to compensate for the previous inclusion: at the isolated points, the chain rule would produce a strange result that does not reflect the differential inclusion and so the chain rule is allowed to diverge back to the true result. The

Iterate Assumptions for Stochastic Subgradient Descent involve 'almost surely' twice. The first instance is used to prevent functions that tend to infinity from being excluded by allowing for the 0% possibility that the iterates do reach infinite values. The second instance is used to permit a few error terms $\xi_{\mathbf{k}}$ that are biased and/or unbounded while ensuring that their quantity is insignificant enough to not impact the overall convergence of the algorithm. These inclusions flow on to the Lemma Almost Sure Implication of Iterate Assumptions, the Convergence of Stochastic Subgradient Descent Under Practical Assumptions Theorem, and the corollaries Convergence of Stochastic Subgradient Descent on Regular Functions and Convergence of Stochastic Subgradient Descent on Tame Functions.

Overall, we can see that the presence of 'almost everywhere' or 'almost surely' does impact various definitions and results. However, these impacts are limited, most notably in Rademacher's Theorem which does not diminish the significance of Stochastic Subgradient Descent.

3.3.2 Limits of Assumptions

The Iterate Assumptions for Stochastic Subgradient Descent and Lyapunov Assumptions for Stochastic Subgradient Descent establish a set of conditions for a function to converge under Stochastic Subgradient Descent. We then outlined two classes of functions that satisfy both sets of assumptions: Regular Functions and Whitney Stratifiable Functions. This is very helpful for practically applying Stochastic Subgradient Descent. However, these classes only provide a partial picture of the conditions imposed by the assumptions. To better illustrate the nature of these conditions, we now examine a handful of examples, some of which are satisfactory, some of which are not.

We begin by recalling a previous example set out in Equations (3.1.1) and (3.1.2). As identified at the time, although this function has an infinite amount of non-differentiable points, it is still locally Lipschitz and so it is a viable candidate for applying Stochastic Subgradient Descent. However, a few properties of this function suggest that it may not satisfy the assumptions: firstly, as x tends towards infinity, the function does as well; secondly, there are infinitely many non-differentiable points around 0; and thirdly, it is defined as the infinite sum of piecewise linear functions. Despite these difficult properties, this function does satisfy the assumptions: the first property may seem to conflict with assumption 2 of the Iterate Assumptions for Stochastic Subgradient Descent, but as long as the initial point is not infinity, then all future iterates will also be bounded; the second property suggests that the first assumption of Lyapunov Assumptions for Stochastic Subgradient Descent may not hold, but the non-differentiable points are only countable and also are non-critical, preventing any exclusion; and finally, although the third property shows that the function is not semialgebraic, it is Whitney C^1-stratifiable and so all the Lyapunov Assumptions for Stochastic Subgradient Descent hold. Therefore, this strange infinitely piecewise function does satisfy all assumptions and can be optimised with Stochastic Subgradient Descent.

Our second example is as follows:

$$f_n(x) = \max\{1 - n|x|, 0\}$$

Figure 3.10 illustrates a few instances for various values of n. For any $n \in \mathbb{N}$, this

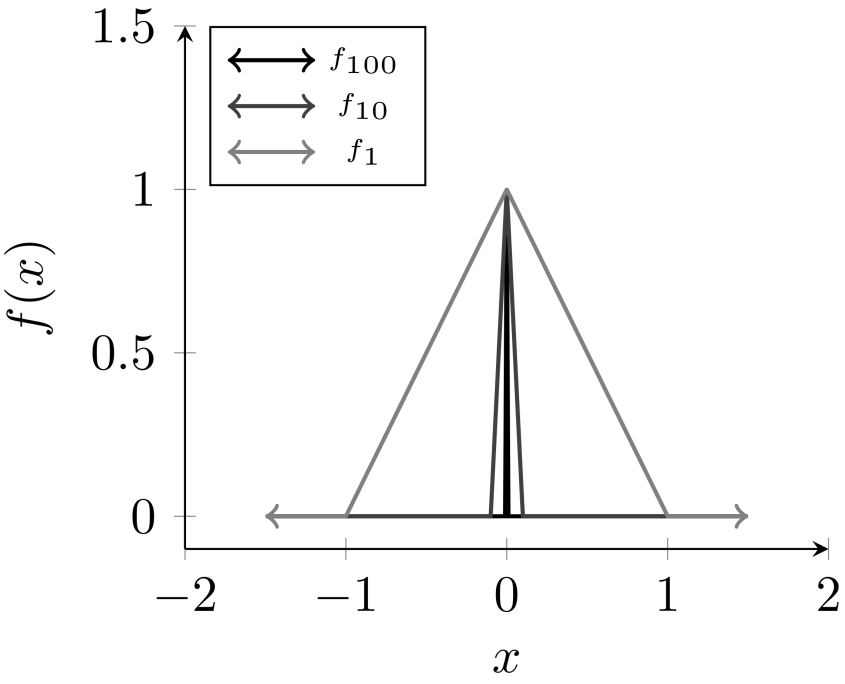


Figure 3.10: f_n for $n = 1, 10, 100$

f_n satisfies the assumptions. This is expected: the function appears to be very well behaved. If we take the limit as n goes to infinity, we get the following function:

$$f_\infty(x) = \begin{cases} 1 & x = 0 \\ 0 & \text{otherwise} \end{cases}$$

This function is not continuous, and so not locally Lipschitz. Somewhat surprisingly though, that is the only issue with the function: it satisfies all other assumptions. In particular, it satisfies the Lyapunov Assumptions for Stochastic Subgradient Descent: the only critical values are 0 and 1 and so the non-critical values are dense in $\mathbb{R}$; and, all points are critical and so the second assumption is vacuously true. This illustrates exactly how permissive the assumptions are.

We now consider a classic counterexample from analysis: the Weierstrass function. It is defined on the domain $[0, 1]$ as

$$\mathcal{W}_{a,b}(x) = \sum_{n=0}^{\infty} a^n \cos(b^n \pi x)$$

for $0 < a < 1$, b a positive odd integer, and $ab > 1 + \frac{3}{2}\pi$. This function is continuous, but not Lipschitz continuous. Furthermore, the Weierstrass function is nowhere differentiable and its Clarke Subdifferential is:

$$\partial \mathcal{W}_{a,b}(x) = \mathbb{R}$$

for all $x \in [0,1]$[5] [101]. This is a strange result: at all points, the Clarke Subdifferential is the entire number line. This immediately violates the first assumption of Lyapunov Assumptions for Stochastic Subgradient Descent, but the second assumption is also violated as the direction of the path is entirely randomised based on what specific subdifferential is sampled from $\mathbb{R}$. Thus, the Weierstrass function is indeed pathological and certainly can not be minimised using Stochastic Subgradient Descent.

Finally, we examine the following theorem from 'Lipschitz Functions with Prescribed Derivatives and Subderivatives' by J. Borwein et Al. [21]:

Theorem 3.8 (Locally Lipschitz Function with Prescribed Subdifferentials)**.** *Let $\alpha, \beta : \mathbb{R} \to \mathbb{R}$ be continuous functions defined on an open interval (a,b). Then there exists a locally Lipschitz function $f : \mathbb{R} \to \mathbb{R}$ such that $\partial f(x) = [\alpha(x), \beta(x)]$ for all $x \in (a,b)$.*

This theorem states that a one-dimensional locally Lipschitz function with any subdifferentials can always be constructed. As such, many locally Lipschitz functions exist that do not satisfy the Iterate Assumptions for Stochastic Subgradient Descent and Lyapunov Assumptions for Stochastic Subgradient Descent. However, the vast majority of these functions are pathological and so are not relevant to practical application. Therefore, while this theorem outlines a large class of functions that do not satisfy the Iterate Assumptions for Stochastic Subgradient Descent and Lyapunov Assumptions for Stochastic Subgradient Descent, it is more illustrative of the limits of the assumptions than an argument against Stochastic Subgradient Descent and other descent methods.

3.3.3 Computability of the Clarke Subdifferential

In our analysis so far, we have only considered the Clarke Subdifferential as a mathematical object. We have heeded no attention to its practical use and implementation. Unfortunately, the definition of the Clarke Subdifferential is not a constructionist one: that is, the definition does not provide a concrete method for calculating the subdifferential. The equivalent definition given by Clarke Subdifferential by Directional Derivatives is also not constructionist and so of no assistance. As a result, the computability and practical implementation of the Clarke Subdifferential is not resolved by the mathematical theory.

[5] This result does not conflict with the Properties of the Clarke Subdifferential, specifically $\partial f(\mathbf{x})$ being a compact set, as those properties only apply for locally Lipschitz functions, which the Weierstrass function is not.

The monograph 'Minimization Methods for Non-Differentiable Functions' by N. Shor [91] outlines a method for calculating a single element of the Clarke Subdifferential with a computation complexity similar to calculating a standard gradient. This seems promising, but the method only applies to convex functions, which limits the application of Stochastic Subgradient Descent to a class even more limited than the Regular Functions. Shor also goes on to show that there is no universal algorithm that produces an approximate subdifferential g such that

$$\min_{g' \in \partial f(\mathbf{x})} \left\| g - g' \right\|_2 < \varepsilon$$

for any $\varepsilon > 0$ and function f. Given this, it would seem that the Clarke Subdifferential can not be used practically. However, Shor does prove that a relaxed approximation is possible:

Theorem 3.9 (Construction of Approximate Clarke Subdifferential)**.** *([91], Chp. 1, Sec. 3) Let $f : \mathbb{R}^d \to \mathbb{R}$ be a locally Lipschitz function and $x \in \mathbb{R}^d$. Let $\varepsilon, \delta > 0$. There exists an algorithm for producing a vector $\mathbf{g}$ such that:*

$$\min_{g' \in \partial f(\mathbf{x_0})} \left\| g - g' \right\|_2 < \varepsilon$$

for some $x_0 \in B_\delta(x)$.

This theorem states that we can produce a close approximation g of the Clarke Subdifferential at a point close to the actual point: there is a stable[6] algorithm for the Clarke Subdifferential. This is a strong and encouraging result: few functions have algorithms that produce approximations beyond stability. Furthermore, Shor proves the Construction of Approximate Clarke Subdifferential Theorem by explicitly constructing the approximation g. We could use this construction for producing an approximation of the Clarke Subdifferential, but unfortunately the method has a high computational complexity. Additionally, it only produces one value of the Clarke Subdifferential, not the entire set, which is problematic for checking the stopping criteria $0 \in -\partial f(\mathbf{x})$. Thus, for practical applications, we must use alternative methods for practically implementing the Clarke Subdifferential.

The most prevalent method used in applications is a sampling approach. Given that a locally Lipschitz function is differentiable almost everywhere, we can safely sample

[6] In the mathematical sense: Given a function a, an algorithm $\tilde{a}$ for that function is stable if, for all data $\mathbf{x} \in X$ we have

$$\frac{\|\tilde{a}(\mathbf{x}) - f(\tilde{\mathbf{x}})\|_2}{\left\|\tilde{f}(\tilde{\mathbf{x}})\right\|_2} \leq c_1 \varepsilon$$

for $c_1 > 0$ and some $\tilde{x}$ with

$$\frac{\|\tilde{\mathbf{x}} - \mathbf{x}\|_2}{\|\mathbf{x}\|_2} \leq c_2 \varepsilon$$

for $c_2 > 0$

nearby points and calculate their gradient to approximate the Clarke Subdifferential:

$$\partial f(\mathbf{x}) \approx \operatorname{conv}\{\nabla f(\mathbf{x_i}) : \forall i = 1, \ldots, k, \mathbf{x_i} \in B_r(\mathbf{x})\} \tag{3.3.1}$$

for some small radius $r \in \mathbb{R}$ and sampling quantity $k \in \mathbb{N}$. This method was pioneered by J. Burke et Al. in their paper 'Approximating Subdifferentials by Random Sampling of Gradients' [26]. This method is the most prevalent for three reasons: firstly, the intuition is simple: the limiting process in the definition is approximated by sampling from the local neighbourhood; secondly, this intuition translates well into mathematical proof, as the approximation converges to the true Clarke Subdifferential as r and k limit to 0 and infinity, respectively [26]; and finally, using this sampling method results in the overall Stochastic Subgradient Descent algorithm being very similar to the stochastic gradient sampling descent method, a variant of Stochastic Gradient Descent that seeks to improve robustness by choosing a descent direction from the convex hull of the local neighbourhood's gradients [58, 25]. These similarities can then be used to provide further insight into each algorithm's behaviour and properties. Due to these reasons, this sampling method is the most common practical implementation of Clarke Subdifferential.

There is an entirely different solution to the problem of computing the Clarke Subdifferential. Some machine learning models use automatic differentiation to produce an approximation of the gradient of the objective function [14]. However, this method can be modelled as an approximation of the Clarke Subdifferential instead [19, 20]. This reframing often proves that these methods satisfy the conditions for Convergence of Stochastic Subgradient Descent Under Practical Assumptions and thus will solve non-smooth optimisation problems. However, there are two issues with this alternative solution. Firstly, it does not properly resolve the issue of computing the Clarke Subdifferential and so can not be used to produce a practical implementation for Stochastic Subgradient Descent. Any framework or machine learning method without a robust automatic differentiation system is not impacted by this solution. Secondly, automatic differentiation methods vary between implementations and are subject to conditions external to the input parameters of Stochastic Subgradient Descent. Therefore, it can often be difficult to mathematically model the entire process and verify that it does indeed satisfy the conditions for Convergence of Stochastic Subgradient Descent Under Practical Assumptions. Due to these two reasons, general methods for practical implementing the Clarke Subdifferential, such as the sampling method, are still important and worth pursuing.

Despite the shortcomings of the automatic differentiation method for approximating the Clarke Subdifferential, it explains why Applying Smooth Optimisation Techniques to Machine Learning has been surprisingly resilient in the face of the critical issues that arise when applying it to non-smooth models. Approximating a gradient with automatic differentiation can make Stochastic Gradient Descent more robust to non-smooth objective functions if the method models some approximation of the Clarke Subdifferential. So the popularity of machine learning frameworks that use such methods, for

example, PyTorch and TensorFlow [75, 2], have accidentally mitigated the critical issues somewhat. This does not mean that all models produced by these frameworks are safe from the critical issues, but merely that these models may have better robustness and performance, due to a higher chance of convergence, than models trained via other methods.

3.3.4 Application to Machine Learning

As set out in the subsection Applying Smooth Optimisation Techniques to Machine Learning, our primary purpose for developing theory for non-smooth optimisation is to ensure that machine learning models can be properly optimised. Specifically, our goals are:

1. An extension of Stochastic Gradient Descent that can be applied to non-smooth machine learning models; and,

2. Convergence theory for that extension that is encompassing enough that most, if not all, non-smooth machine learning models can be safely trained to convergence.

The Stochastic Subgradient Descent algorithm certainly succeeds in achieving the first goal. Non-smooth machine learning methods must have continuous objective functions, so we only need to check that they are locally Lipschitz. Fortunately, as the models have finite scale, they must be locally Lipschitz, unless a component is explicitly defined to not be locally Lipschitz. For example, all machine learning models outlined in the subsection Mathematical Representations of Machine Learning Models are locally Lipschitz, unless purposefully setup otherwise, e.g. the loss function is is defined to be $\cos\left(\frac{1}{x}\right)$. Therefore, Stochastic Subgradient Descent can be applied to most, if not all, practical implementations of non-smooth machine learning models.

Confirming success for the second goal is a more involved process. To do so, we must verify that a machine learning model satisfies the Iterate Assumptions for Stochastic Subgradient Descent and Lyapunov Assumptions for Stochastic Subgradient Descent, and therefore converges under the Convergence of Stochastic Subgradient Descent Under Practical Assumptions Theorem. Using these assumptions and theorem alone, it would be a difficult and involved task to prove whether each machine learning method would converge to an optimal point when trained using Stochastic Subgradient Descent. However, we can shortcut this process by instead using Convergence of Stochastic Subgradient Descent on Tame Functions. Generally speaking, we will once again find that machine learning models are Whitney stratifiable unless a component is explicitly defined otherwise. For example, given the general representation of a supervised machine learning in Equation (2.3.1), we can produce the following novel corollary:

Corollary 3.3 (Application of Stochastic Subgradient Descent to Supervised Learning Models)**.** *Let $\mathcal{M}$ be a supervised learning model with representation:*

$$\arg\min_{\mathbf{w}\in\mathcal{W}} \frac{1}{k}\sum_{i=1}^{k} l\left(f\left(\mathbf{x_i};\mathbf{w}\right),\mathbf{y_i}\right) + \lambda r\left(\mathbf{w}\right)$$

where $\mathcal{W} \subseteq \mathbb{R}^d$ is the space of model parameters, l is a loss function, f is the modelling function, r is a regulariser, $D = \{\mathbf{x_i}, \mathbf{y_i}\}_{i=0}^{k}$ is the training data set, and λ is a balancing hyperparameter.

If f, l and r are locally Lipschitz, then Stochastic Subgradient Descent can be applied to the problem.

Furthermore, if f, l, and r are also Whitney C^d-stratifiable and the application satisfies Iterate Assumptions for Stochastic Subgradient Descent, then almost surely the parameter iterates $\{\mathbf{w}_j\}_{j\geq 0}$ will converge to a critical point and the model's regularised empirical risk will converge.

From this corollary, we can immediately infer that Polynomial Fitting and Support Vector Machines will always converge when trained using Stochastic Subgradient Descent. Neural Networks will also converge if the activation functions and loss function are locally Lipschitz and Whitney stratifiable. Most activation functions are locally Lipschitz and Whitney stratifiable, e.g. sigmoid, hyperbolic tangent, absolute value, cosine, RelU, leaky RelU, softmax, etc. [8]. The only significant class of activation functions that do not meet the assumptions are step activation functions as they are not even continuous. Similarly, most standard loss functions are also locally Lipschitz and Whitney stratifiable, e.g. square, absolute, Huber, log-cosh, quantile, reconstruction, entropy, etc. [100]. Therefore, the vast majority of Neural Networks will converge when trained with Stochastic Subgradient Descent. Finally, Deep Declarative Networks are similar to Neural Networks: they will converge if the activation functions, loss function, and also the objective functions of the declarative nodes are all locally Lipschitz and Whitney stratifiable. Given that the declarative nodes must optimise their own objective function to produce an output, it is reasonable to assume that they will be locally Lipschitz and Whitney stratifiable to ensure that the model can even be computed. If the declarative nodes employ constrained optimisation problems, then we must also require that those constraint sets are Whitney stratifiable. Therefore, we find that Deep Declarative Networks will converge under the same conditions as Neural Networks.

3.3.5 Summary of Theory

We began this chapter by illustrating that standard smooth optimisation techniques, most particularly Stochastic Gradient Descent and its variants, are insufficient for training machine learning models: the non-smooth components prevent a guarantee of convergence and can even crash the training regime. We identified that this was a significant

issue that required immediate resolution. Thus, we had two goals: outline an algorithm that can be applied to non-smooth machine learning methods; and, demonstrate that the algorithm will converge for these machine learning applications.

Overall, we are successful: Stochastic Subgradient Descent overcomes the shortcomings of Stochastic Gradient Descent by being applicable to non-smooth machine learning methods, and its convergence theory is sufficiently strong to ensure that almost all practical models will converge to an optimal point. Hence, we have completed our theoretical consideration of non-smooth optimisation and Stochastic Subgradient Descent.

Chapter 4

Application of Theory

So far, we have only considered optimisation and its applications as mathematical problems. In this chapter, we examine the practice of these problems. We begin by implementing the first completely generic version of the Stochastic Subgradient Descent and use this implementation to verify the algorithm's convergence on a variety of functions and problems. We then identify that the speed of Stochastic Subgradient Descent is an important factor that impacts its viability as a practical method and so conduct further experiments to estimate the rate and order of convergence for Stochastic Subgradient Descent. These experiments verify a handful of existing results and also provide direction for future research. The chapter then concludes with a discussion of future pathways for both the theory and practice of non-smooth optimisation.

All experimental code used within this chapter can be accessed at the following GitHub Repository.

4.1 Verifying Convergence

In this section, we outline our methodology and results for the verification experiments. The section opens with a short discussion on our motivation for conducting these experiments, establishing their necessity and our expectations. After this discussion, our first technical step is to provide an overview of our practical yet generic Stochastic Subgradient Descent implementation and the various issues that were overcome to complete the method. We then specify our methodology for the verification experiments, presenting the experimental parameters and variables. An overview of our experimental results are presented and analysed, enabling us to conclude that Convergence of Stochastic Subgradient Descent Under Practical Assumptions does translate to the real-world. We then conclude this section by discussing the implications of this translation to training non-smooth machine learning models using Stochastic Subgradient Descent.

4.1.1 Motivation for Verification

The primary goal of the verification experiments is to practically confirm the theory outlined in the previous chapter, in particular the Convergence of Stochastic Subgradient Descent Under Practical Assumptions. However, given that this theory is backed by mathematical proof, it is worth questioning the necessity of such experiments.

The primary reason is to verify the translation of theory from the mathematical domain to the real-world. The mathematical domain is entirely theoretical and only operates upon the rigid axioms established at the core of the logical system. We can construct mathematical models that reflect aspects and properties of the real-world, however, they are still merely models. Thus, mathematical results only describe the behaviour of mathematical models, not the precise nature of their real-world inspirations. Therefore, practical experiments are needed to verify the translation of a mathematical result to the real-world.

In our situation, there are two translations that we must verify. Firstly, we must verify that the mathematical definition of the Stochastic Subgradient Descent algorithm can be practically implemented in the real-world. As our discussion of the Computability of the Clarke Subdifferential illustrates, this translation is not immediate nor easy, and so we must examine the technical techniques needed complete a practical implementation. The next subsection Implementation covers this examination. Secondly, we must verify the translation of the theory of Stochastic Subgradient Descent, in particular the Convergence of Stochastic Subgradient Descent Under Practical Assumptions, to the real-world. Despite our best efforts, it is possible that the derived results can not be applied to any real-world application of Stochastic Subgradient Descent, and so convergence may not be guaranteed. The experiments and results outlined in this section will seek to verify this translation.

The necessity to verify these translations is magnified by the apparent absence of any generic implementation of Stochastic Subgradient Descent and thus any comprehensive test of Convergence of Stochastic Subgradient Descent Under Practical Assumptions. There exists a handful of implementations of Stochastic Subgradient Descent [11, 61], however, each is limited to a specific domain, e.g. support vector machines. Thus, there is no generic, practical implementation of Stochastic Subgradient Descent, as far as we are aware. Hence, the translation of both the algorithm and theory is important to verify in the general settings.

4.1.2 Implementation

When we set out the formal algorithm for Stochastic Subgradient Descent, we noted that its similarities to Stochastic Gradient Descent meant that implementations would be ‘almost identical’. As a result, the majority of our Stochastic Subgradient Descent implementation is straightforward: the methods and techniques for implementing such an algorithm are well established, most particularly in the various extensions of Stochas-

tic Gradient Descent [105, 107, 78, 57]. In fact, there is only one line of the algorithm that required attention: the sampling of an approximate subdifferential.

As outlined when we considered its computational issues, the most common method for approximating the Clarke Subdifferential is the sampling approach defined by Equation (3.3.1) [26]. For the reasons discussed before in Subsubsection Computability of the Clarke Subdifferential, as well as its prevalence and subsequent wide acceptance, we used this method for our implementation. However, the method is not an algorithm for practical calculation, but rather another mathematical relation. Furthermore, the paper which pioneered it does not consider any issues regarding practice, application, or implementation, and all following papers build solely upon the theory. Thus, we were required to derive a novel algorithm for practically calculating the approximation:

Algorithm 5: Approximating the Clarke Subdifferential

input : Locally Lipschitz function $f : \mathbb{R}^d \to \mathbb{R}$. Point $\mathbf{x} \in \mathbb{R}^d$. Neighbourhood radius parameter $r \geq 0$. Sampling parameter $N \in \mathbb{N}$

output: Approximation of the Clarke Subdifferential as given in Equation (3.3.1).

Sample gradients from d-sphere of radius r around $\mathbf{x}$;
for $k = 1, 2, \ldots, N$ **do**
    Sample a point;
    $\mathbf{p_k} \leftarrow S^d(r) + \mathbf{x}$;
    Calculate gradient at sampled point;
    $\mathbf{g_k} \leftarrow \nabla f(\mathbf{p_k})$;
Compute convex hull;
$A \leftarrow \text{conv}\{\mathbf{g_k} : \forall k = 1, \ldots, N\}$;
return *A;*

This algorithm immediately satisfies the mathematical definition of the sampling approach given by Equation (3.3.1) and so we can be sure that it is indeed an approximation of the Clarke Subdifferential. Technically, both the convex hull and d-sphere of radius r, $S^d(r)$, are mathematical constructs that are also impossible to compute exactly within continuous domains. However, their ubiquity within computational mathematics has led to many accurate approximations being derived [10, 99]. For the convex hull, we elected to use the convex hull approximation method within SciPy [98], which is dependent upon the popular `quickhull` method used for computer graphics [13]. For sampling from the d-sphere, we employed a fast yet stable method using normal distributions [69].

Finally, as noted previously in our original discussion of Stochastic Subgradient Descent, any heuristic, either deterministic or stochastic, can be used to sample from the Clarke Subdifferential. Although we are not investigating and analysing the consequences of such heuristics, our implementation does follow the mathematical algorithm and allows for any heuristic to be used. For our experiments, we followed our previous commitment and only used random uniform sampling.

Overall, we produced a practical and generic algorithm for Stochastic Subgradient Descent that is fast yet simple and back by rigorous theory. As far as we are aware, this algorithm is a first of its kind.

4.1.3 Verification Methodology

Our next aim is to verify the translation of the Convergence of Stochastic Subgradient Descent Under Practical Assumptions to the real-world. This can be achieved very directly: we can apply Stochastic Subgradient Descent to a variety of non-smooth optimisation problems and analyse the results to determine if the iterates converge to a local minimum. While obvious and simple, this verification method covers all possible issues: firstly, we can determine whether the Iterate Assumptions for Stochastic Subgradient Descent and Lyapunov Assumptions for Stochastic Subgradient Descent can be feasibly achieved in practical settings with our choice of optimisation problems and hyper-parameters for Stochastic Subgradient Descent; and secondly, we can determine whether an application with these assumptions satisfied does indeed converge. Thus, we must establish a methodology for applying Stochastic Subgradient Descent to a variety of non-smooth optimisation problems.

We begin by selecting functions that we believe will satisfy the Lyapunov Assumptions for Stochastic Subgradient Descent. Fortunately, this is a simple process, as the two Lemmas Chain Rule under Statifications and Critical Values under Stratifications demonstrate that any locally Lipschitz function that admits a Whitney C^d-stratification will satisfy both of the Lyapunov Assumptions for Stochastic Subgradient Descent. Furthermore, we know that any semialgebraic function is Whitney C^d stratifiable. Therefore, there is a very wide class of functions to select from. We used the following:

$$\begin{aligned}
\text{Quadratic Function}: &\quad ax^2+bx+c \\
&\quad a\in(0,1), b\in(-10,10), c\in(-10,10) \\
\text{ReLU}: &\quad \begin{cases} 0 & x<0 \\ x & x\geq 0 \end{cases} \\
\text{Scaled } p-\text{Norms}: &\quad a\left\|\mathbf{x}\right\|_p = a\left(\sum_{i=1}^{d}|x_i|^p\right)^{\frac{1}{p}} \\
&\quad p\in\{1,2,\infty\}, a\in(0,1)
\end{aligned} \tag{4.1.1}$$

$$p - \text{Norm Waves} : \qquad a \cos\left(\frac{\|\mathbf{x}\|_p}{b}\right)$$
$$p \in \{1, 2, \infty\}, a \in (1, 2), b \in (1, 2)$$

$$p - \text{Norm Subtracting } p - \text{Norms Waves} : \qquad c\|\mathbf{x}\|_p - a \cos\left(\frac{\|\mathbf{x}\|_p}{b}\right)$$
$$p \in \{1, 2, \infty\}, a \in (1, 2), b \in (1, 2), c \in (0, 2)$$

For the parameters a, b, and c, a value was randomly sampled for each test, while we tested each of the three values for p independently.

We used these particular functions for three reasons. Firstly and most importantly, they are all locally Lipschitz and semialgebraic, and so will test the translation of all the lemmas and assumptions mentioned above. Secondly, many of them are commonly used within machine learning: ReLU is primarily used as an activation function with neural networks [81]; p-norms are often used within machine learning as regularisers [95]; and, cos and other trigonometric functions are used for many statistics based model and loss functions, such as entropy or likelihood functions [100]. Finally, the functions cover a wide variety of mathematical settings and so allow us to verify the translation of Stochastic Subgradient Descent across many domains: the quadratic function and 2-norm are smooth, testing that Stochastic Subgradient Descent simplifies to Stochastic Gradient Descent in the smooth setting in both 1 and higher dimensions; ReLU and the 1 and infinity norms are non-smooth and so test the fundamental application of Stochastic Subgradient Descent to non-smooth functions, again in 1 and higher dimensions; the waves are a mix of smooth and non-smooth functions with many (equally good) global minima, testing which minimum Stochastic Subgradient Descent converges to; and finally, the waves on p-norms have many local minima but only one global minimum, testing whether Stochastic Subgradient Descent converges to local minima, as we would expect, or exhibits hitherto unknown global convergence properties. Overall, the test functions were selected to verify the translation of the Lyapunov Assumptions for Stochastic Subgradient Descent to the real world in a variety of settings relevant to machine learning.

The next step is to implement a testing environment for applying Stochastic Subgradient Descent that will satisfy the Iterate Assumptions for Stochastic Subgradient Descent. The functionality of the testing environment is very simple - just apply Stochastic Subgradient Descent and record the iterates and their function values - however three parameters will need to be carefully selected: the step size sequence $\{\alpha_k\}_{k\geq 0}$; the initial points $\mathbf{x_0}$; and, the error sequence $\{\xi_\mathbf{k}\}_{k\geq 0}$. We can consider each independently, as the assumptions with the Iterate Assumptions for Stochastic Subgradient Descent are not inter-dependent. Thus, we can determine each in turn.

The first assumption of the Iterate Assumptions for Stochastic Subgradient Descent

imposes three conditions upon the step size sequences:

$$\alpha_k \geq 0 \qquad \sum_{k\geq 0} \alpha_k = \infty \qquad \sum_{k\geq 0} {\alpha_k}^2 < \infty$$

i.e. non-negative, not summable, but square summable. We can construct a family of such sequences from two known properties. Firstly, we recalled a result from analysis that the sum

$$\sum_{n\in\mathbb{N}} \left(\frac{1}{n}\right)^r$$

converges when $r > 1$ and diverges otherwise [96]. Thus, we can infer that the same sum is not summable, but is square summable when $\frac{1}{2} < r \leq 1$. Secondly, any finite sequence can be added to the start of a series without impacting its convergence:

$$\left(\sum_{k=0}^{n-1} \beta_k\right) + \left(\sum_{k=n}^{\infty} \alpha_{k-n}\right) = \infty \iff \sum_{k\geq 0} \alpha_k = \infty$$

Given these two results, we know that the family of sequences with terms defined by

$$\alpha_k = \begin{cases} \beta_k & 0 \leq k \leq n \\ \left(\frac{1}{k-n}\right)^r & k > n \end{cases} \tag{4.1.2}$$

for $\frac{1}{2} < r \leq 1$ and any finite, non-negative sequence $\{\beta_k\}_{0\leq k\leq n}$ will satisfy the three conditions the first assumption of the Iterate Assumptions for Stochastic Subgradient Descent imposed upon the step size sequence. Many sequences outside of this family satisfy the three conditions, however, we opted to use this family for our verification tests. We did this for two reasons. Firstly, the initial $n + 1$ β_k terms provide a lot of flexibility that can be tailored to specific situations. For example, if the domain space is large, then these β_k values can be set to a high constant value, enabling the iterates to quickly move towards a local minimum before the step size is successively decreased. Secondly, the family is very simple and so verification of the translation ought to be direct and immediate.

The second assumption of the Iterate Assumptions for Stochastic Subgradient Descent imposes a condition upon the iterates: almost surely, the iterates bounded:

$$\sup_{k\geq 0} \|\mathbf{x_k}\|_2 < \infty$$

Fortunately, none of our chosen functions tend towards negative infinity as the norm of the iterates grows, i.e.

$$\|\mathbf{x_k}\|_2 \to \infty \not\Rightarrow f(\mathbf{x_k}) \to -\infty$$

Therefore, as the algorithm is applied as the iterates move 'downhill', they will not tend towards infinity themselves and so will remain bounded. As a result, we can set our

initial iterate value $\mathbf{x_0}$ to be anything. For ease, we uniformly sampled the initial iterate from $[-10, 10]^d \setminus (-5, 5)^d$, i.e. each dimension is sampled from $[-10, -5] \cup [5, 10]$.

The final assumption of the Lyapunov Assumptions for Stochastic Subgradient Descent imposes conditions upon the error sequence $\{\xi_\mathbf{k}\}_{k\geq 0}$: roughly, the errors must be unbiased, uncorrelated, and bounded. This assumption is predominantly satisfied by implementing the algorithm and testing environment on a computer: computational errors, from rounding errors, memory disturbances, bit-adder operations, and more, are naturally uncorrelated and bounded [112]. Most computational errors are also unbiased, but certain methods for rounding can be biased, for example, some always round up [104]. To bypass this issue, we manually set a lower precision of $1\text{e}-6$, i.e. just below single precision [1], and manually checked if an iterate would round to a non-differentiable point. Therefore, with this small change, the testing framework also satisfies the final assumption of the Lyapunov Assumptions for Stochastic Subgradient Descent.

We do note that the lower precision implies a limit of 1 million steps, as $\frac{1}{1{,}000{,}000} = 1\text{e}-6$. The convergence theory assumes that the algorithm will continue forever, letting k tend to infinity. This is impossible within the real-world, however, this is only needed as real numbers have infinite precision. Given that we are operating with finite precision, we will only need to verify convergence within that precision. Thus, the verification test is still valid. Furthermore, a limit of 1 million steps is a reasonable bound that reflects real-world applications.

To summarise our methodology, we aim to verify the translation of the Convergence of Stochastic Subgradient Descent Under Practical Assumptions to the real-world by testing the application of Stochastic Subgradient Descent. We have chosen to conduct this test on the objective functions list in Equation (4.1.1), with the step size sequence given by Equation (4.1.2), an initial iterate randomly sampled within $[-10, 10]^d \setminus (-5, 5)^d$, a precision of $1e-6$, and a maximum of 1 million steps. These choices satisfy the Iterate Assumptions for Stochastic Subgradient Descent and Iterate Assumptions for Stochastic Subgradient Descent, while also covering a wide array of common applications, and thus will enable us to verify the viability of using Stochastic Subgradient Descent to solve non-smooth optimisation problems.

Finally, we recall that Stochastic Subgradient Descent has stochastic elements. So even if we did not randomise the parameters for the objective functions, the experiments are not deterministic. Thus, it is worth repeating the runs to analyse the variability of the algorithm. 10 repetitions for each objective function, with the parameters randomised for each, will be sufficient to conduct this analysis.

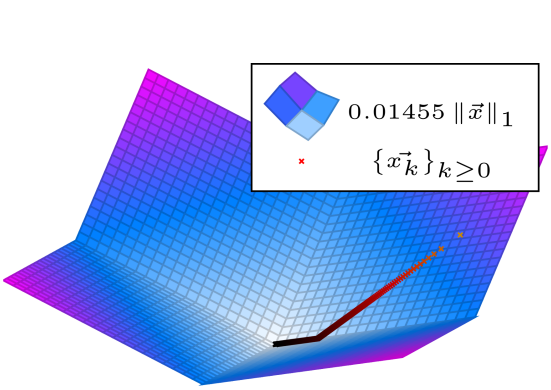
4.1.4 Verification Results

We now analyse the results from our experiments verifying the viability of using Stochastic Subgradient Descent to solve non-smooth optimisation problems. As 110 total runs were completed, we do not report and analyse each individually. Instead, we summarise

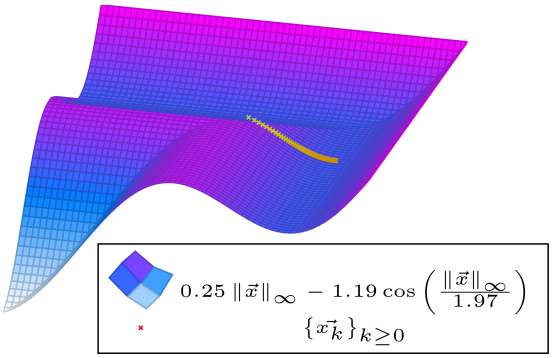
the results and assess the overall trends. A complete record of the experiments and the results can be found in Appendix Experimental Results.

Runs fell into one of two groups: either the algorithm halted before the 1 million step limit, or the algorithm reached this limit. We consider each group separately.

We know that the stopping condition for Stochastic Subgradient Descent is that $0 \in -\partial f(\mathbf{x_k})$, i.e. the current iterate is a Clarke critical point. Therefore, for the first group of runs that halted before the step limit, we know that the algorithm reached a Clarke critical point. This strongly suggests that the application of Stochastic Subgradient Descent was successful, but it is possible that the discovery of a Clarke critical point was accidental. To disprove this possibility, we can examine the path of the iterates and the function values. For any run that halted, we find that the iterates follow a clear trajectory along the function in a downhill direction towards a local minimum:



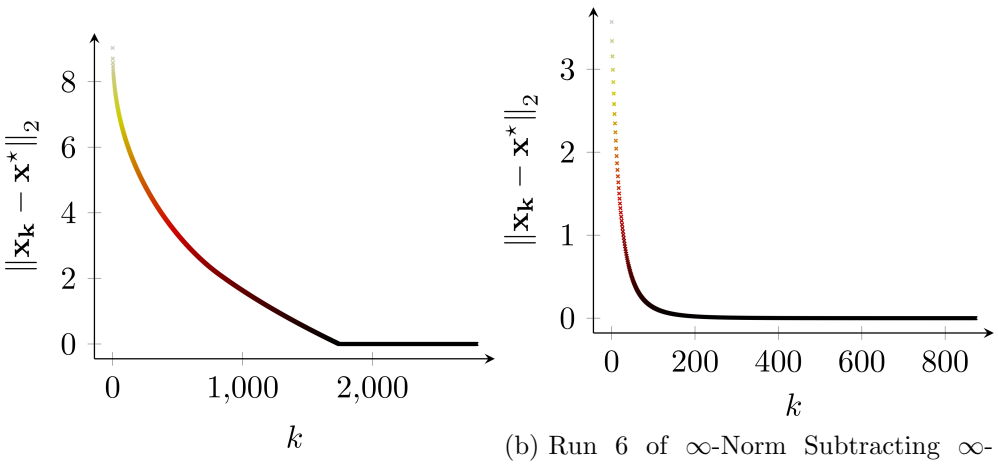
(a) Run 5 of Scaled 1-Norm



(b) Run 6 of ∞-Norm Subtracting ∞-Norm Waves

Figure 4.1: Converging Iterates for Halting Runs

Note that the path of the iterates on the scaled 1-norm follow non-smooth points and the path on the ∞-norm subtracting ∞-norm waves crosses non-smooth points, evidencing the robustness of Stochastic Subgradient Descent to non-smooth elements. In fact, many runs, especially those involving 1 and ∞-norms, followed non-smooth points downhill. In general, this downhill trajectory is the exact behaviour we would expect: the Convergence of Stochastic Subgradient Descent Under Practical Assumptions showed that the iterates will eventually follow such a trajectory and thus converge. Finally, we can also plot the error between the iterates and the final iterate $\mathbf{x}^\star$ to confirm this convergence:



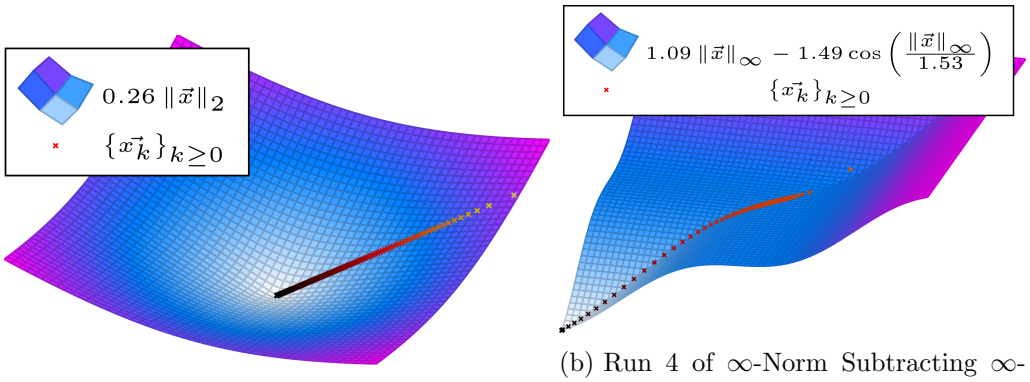
(a) Run 5 of Scaled 1-Norm

(b) Run 6 of ∞-Norm Subtracting ∞-Norm Waves

Figure 4.2: Convergence of Iterate Errors for Halting Runs

Given these results, we have strong evidence that the discovery of a Clarke critical point and the halting of Stochastic Subgradient Descent was no accident. Therefore, this group of runs provides experimental evidence that the Convergence of Stochastic Subgradient Descent Under Practical Assumptions translates to real-world application.

The second group of runs is more interesting. The runs did not halt and so at no point did the algorithm find a Clarke critical point. Alone, this may suggest that the application of Stochastic Subgradient Descent was unsuccessful and therefore at least one of the Iterate Assumptions for Stochastic Subgradient Descent, Lyapunov Assumptions for Stochastic Subgradient Descent, and Convergence of Stochastic Subgradient Descent Under Practical Assumptions do not translate to the real-world. However, when we examine the iterates, we can see that these non-halting runs exhibit the same behaviour as the halting runs from the first group:

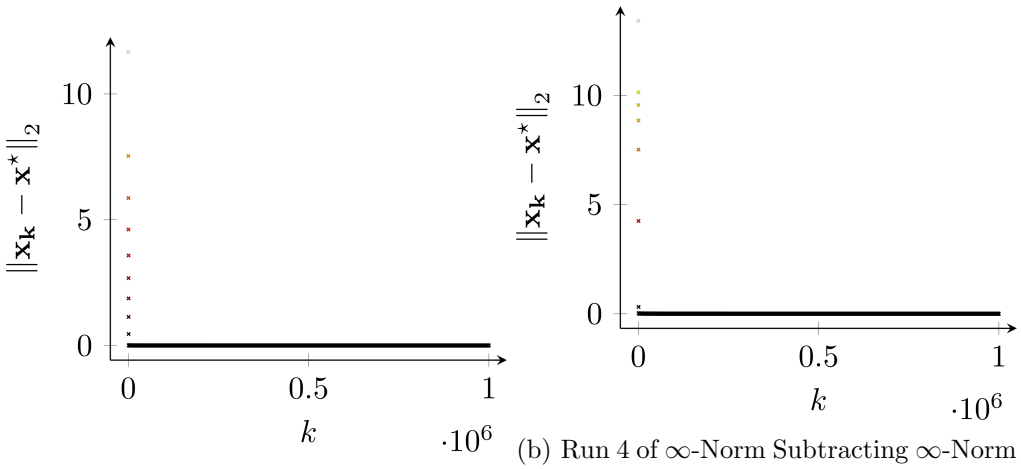


(a) Run 1 of Scaled 2-Norm

(b) Run 4 of ∞-Norm Subtracting ∞-Norm Waves

Figure 4.3: Converging Iterates for Non-Halting Runs

The iterates follow a clear downhill trajectory, tending to and undisrupted by non-smooth points, as they move towards a local minimum. Similarly, the error in the iterates also decreases and converges to a locally optimal value:



(a) Run 1 of Scaled 2-Norm

(b) Run 4 of ∞-Norm Subtracting ∞-Norm Waves

Figure 4.4: Converging Iterates Errors for Non-Halting Runs

This behaviour suggests that these runs were still successful applications of Stochastic Subgradient Descent, but were halted before convergence.

This hypothesis is further supported by which runs and objective functions halted and which did not. There were two groups of runs that did not halt. The first group very quickly reached the local neighbourhood of an optimal point, but then bounced around, never settling to the local minimum. The runs shown in Figures 4.3 and 4.4 fall into this group. This issue is most likely caused by step sizes that are too large: Stochastic Subgradient Descent can not move 'downhill' and achieve an optimal point if the step sizes overshoot the lower regions of the function. Thus, while reducing the step sizes would slow the initial speed of the convergence, it would eliminate the overshooting issue and so allow the algorithm to reach a Clarke critical point. The second group of runs that did not halt never even reached the local neighbourhood of a Clarke critical point. So Stochastic Subgradient Descent could not travel far enough to converge. Within a finite number of steps, the distance traveled is determined by the local Lipschitz constant of the function, i.e. local slope, and the step size sequence. Many runs in this group were on the scaled p-norms, which have only one Clarke critical point at $\mathbf{0}$ and also bounded Lipschitz constants, supporting our explanation for why this group of runs failed. Given that the runs were still tending towards an optimal point, but failed to travel far enough, we could increase the step sizes to solve this problem and converge. However, first group failed to converge because the step sizes were too large, and so the two groups have conflicting solutions. Thus, we see that the choice of Stochastic Subgradient Descent's hyperparameters becomes significant: the non-halting runs did tend towards a locally

optimal point, but failed to converge due to the choice of hyperparameters. Thus, while these non-halting runs did not converge to a Clarke critical point, their behaviour still suggests that they were tending towards such a point and thus were successful applications of Stochastic Subgradient Descent.

Therefore, while the halting runs are more conclusive, our experimental data from all runs provides significant evidence that the Convergence of Stochastic Subgradient Descent Under Practical Assumptions translates to real-world application.

4.1.5 Conclusion of Verification Experiments

In this section, we have completed two verifications. Firstly, we verified that Stochastic Subgradient Descent can be practically and generically implemented and applied. This was achieved by implementing such a program, representing a first of its kind. Secondly, we have verified that Convergence of Stochastic Subgradient Descent Under Practical Assumptions translates to real-world application, with assumptions that are achievable and results that solve the non-smooth optimisation problem.

Before we leap to conclusions about the application of Stochastic Subgradient Descent to training non-smooth machine learning models, we briefly reiterate two issues that must be noted and considered.

Firstly, we recall that the Optimality Condition for Non-Smooth Optimisation does not imply that every Clarke critical point is a minimum. Instead, this proposition states that, for the objective function of our particular flavour of optimisation problem as given in Equation (2.1.2), any non-Clarke critical point is not an optimal point. For example, the Critical Point Theorem states that any maximum point would also be Clarke critical. Therefore, even though Convergence of Stochastic Subgradient Descent Under Practical Assumptions deals with Clarke critical points, we are relying upon descent methods moving downhill with non-infinitesimal step sizes for the Clarke critical point in question to be a minimum. This situation is deliberate: guaranteeing convergence to an optimal point is impossible without restrictive complexity. For example, if the initial point is a maximum point, then Stochastic Subgradient Descent would immediately halt, satisfied with this Clarke critical point. This could be overcome by adding an assumption about the initial point, but similar issues apply to Clarke critical inflection points[1] which could only be overcome with a far more sophisticated algorithm and very limiting conditions. Thus, the success of Stochastic Subgradient Descent relies upon Convergence of Stochastic Subgradient Descent Under Practical Assumptions being satisfied, its construction as a descent method, and some consideration in its application.

Secondly, in a similar vein, even if Stochastic Subgradient Descent converges to an optimal point, it will not necessarily be globally optimal. As noted in our discussion of

[1] An inflection point is a point where the second derivative changes sign, e.g. $x = 0$ for $f(x) = \sin(x)$. A Clarke critical inflection point is a point that is both a Clarke critical point and an inflection point, e.g. $x = 0$ for $f(x) = x^3$.

Convergence of Stochastic Gradient Descent, finding globally optimal points is sometimes intractable [103] and so we rely upon local optimal points as approximate solutions. In some cases, this is satisfactory, however, some local optimal points may be too shallow for use. In these cases, Stochastic Subgradient Descent must be applied with care and using informed choices of parameters. For example, as seen in our experiments, proximity between the initial point and the minima is important and so initialising the algorithm close to a global optimal point, or at least near deeper local optimal points, will probably lead to better solutions. Additionally, given the presence of stochastic elements, repeated runs of Stochastic Subgradient Descent on the same objective function may produce different results, from which the best optimal point can be selected. As a result, it is often best to divide the available resources between multiple runs to produce a set of candidate solutions, rather than refining a given solution with more precision. Thus, Stochastic Subgradient Descent will converge most often to a local optimal point and sometimes care must be taken to guarantee that the result is sufficient for use.

Both of these issues are relatively minor: in the vast majority of cases, blind application of Stochastic Subgradient Descent will still produce a satisfactory result to the non-smooth optimisation problem. However, they must be considered to properly determine that Stochastic Subgradient Descent can solve non-smooth optimisation problems, and active awareness and management of the issues will also lead to better results.

With the verification complete and final issues resolved, we can make the following conclusion: Stochastic Subgradient Descent can be applied to non-smooth machine learning models, resulting in the model converging to a local optimal. This solves the issue outlined in Applying Smooth Optimisation Techniques to Machine Learning, with Stochastic Subgradient Descent overcoming the shortcomings of Stochastic Gradient Descent and consequently putting the training of machine learning on a solid theoretical, yet practical, basis.

4.2 Rate and Order of Convergence

In this section, we outline our methodology and results for the experiments relating to Stochastic Subgradient Descent's rate and order of convergence. We open by examining what rate and order of convergence are, why it is a significant property of an iterative algorithm, and hence why it is worth determining for Stochastic Subgradient Descent. Given this motivation, we establish a new goal to produce experimental evidence that supports the existing, but limited, theoretical results and can guide future research into the matter. We then develop an experimental methodology for achieving these goals, apply it, and present an overview of the new results. Finally, we summarise these results and analyse what theoretical results they suggest.

Motivation for Rate and Order of Convergence

A theoretical and practical guarantee of convergence opens the door for applying Stochastic Subgradient Descent to train non-smooth machine learning models. However, there are further barriers that could limit this application. One of the most significant is resource requirements: if Stochastic Subgradient Descent is very resource-intensive, it may not be a viable method for solving non-smooth optimisation problems. Many different resource issues could impact the viability of an algorithm, however, one of the most critical is the time requirement, i.e. the algorithm's speed. A horrifically slow algorithm that converges is no more useful than a fast algorithm that may not converge, because within a reasonable time frame neither provides practical guarantees on the accuracy of the solution.

For an iterative algorithm, we can measure speed in two ways: computational steps, i.e. CPU cycles, or iterations. The former is more comprehensive, analysing the number of lines of machine code run throughout the entire algorithm. However, the number of computational steps is also implementation dependent. For example, our generic sampling method for computing the Clarke Subdifferential is going to be slower than an application specific method, as we can not optimise for the application's objective function and the expected iterates. As a result, for an iterative algorithm, the more generic measure of speed is the number of iterations. Formally, this is measured using the following:

Definition 4.1 (Q Convergence)**.** *Suppose we apply an iterative algorithm to produce a sequence of iterates* $\{\mathbf{x_k}\}_{k\geq 0}$ *that limit to* $\mathbf{x}^\star$*. Then the sequence is said to* Converge Q-Linearly *if there exists* $\mu \in (0,1)$ *such that*

$$\lim_{k\to\infty} \frac{\|\mathbf{x_{k+1}} - \mathbf{x}^\star\|}{\|\mathbf{x_k} - \mathbf{x}^\star\|} = \mu$$

In this case, μ *is called the* Rate of Convergence*. If* $\mu = 0$*, then the sequence is said to* Converge Q-Superlinearly*, and if* $\mu = 1$*, then the sequence is said to* Converge Q-Sublinearly*. Finally, the sequence is said to* Converge with Order m *if*

$$\lim_{k\to\infty} \frac{\|\mathbf{x_{k+1}} - \mathbf{x}^\star\|}{\|\mathbf{x_k} - \mathbf{x}^\star\|^m} = \mu \tag{4.2.1}$$

for some non-negative rate of convergence $\mu \geq 0$*, with the additional condition that* $\mu < 1$ *if* $m = 1$*.*

Effectively, the rate of convergence measures the multiplicative relation between successive errors $e_{k+1} = \|\mathbf{x_{k+1}} - \mathbf{x}^\star\|$ and $e_k = \|\mathbf{x_k} - \mathbf{x}^\star\|$, while the order of convergence measure the exponential relation between the successive errors. Note that this means that higher values for the order of convergence are better, but lower values for the rate of convergence are better. Functionally, the rate and order of convergence set an upper limit on how slowly an algorithm will converge. We can use this bound to estimate how

many steps are required to produce a result of a given accuracy, and so the maximum amount of time needed to produce a good solution. Hence, for iterative algorithms, the rate and order of convergence is the critical measure of speed.

If we can mathematically derive Stochastic Subgradient Descent's rate and order of convergence, then we will be able to determine how quickly the algorithm converges and thus whether the algorithm is practically viable. Motivated by this, research has been completed in the area. Older theorems from optimisation can be used to prove that Stochastic Subgradient Descent converges sublinearly in convex settings [71]. Furthermore, the Properties of the Clarke Subdifferential and the relation Lipschitz and Convexity Imply Regularity have recently be used to prove Stochastic Subgradient Descent's sublinear convergence in smooth [45] and weakly convex settings [35].

However, there is a significant issue with the current results. Somewhat surprisingly, it is not the speed. Although it may appear lacklustre given the low order and rate, Stochastic Subgradient Descent is very competitive in comparison to other state-of-the-art optimisation methods. For example, Stochastic Gradient Descent is the current standard for training machine learning models, as outlined in Applying Smooth Optimisation Techniques to Machine Learning, and yet it is a sublinear method with rates of convergence similar to Stochastic Subgradient Descent [22]. Instead, the significant issue lies with the scope of the current results: they only encompass smooth or weakly convex functions. For the smooth case, there is no reason to use Stochastic Subgradient Descent over Stochastic Gradient Descent, and for the weakly convex case, we are limited to a class of functions marginally larger than the Regular Functions. More importantly, neither allows us to assess the speed at which Stochastic Subgradient Descent will train non-smooth machine learning models. Consequently, we need a result that proves Stochastic Subgradient Descent's rate and order of convergence on the class of functions that satisfy the Iterate Assumptions for Stochastic Subgradient Descent and Lyapunov Assumptions for Stochastic Subgradient Descent, or at least on the class of Tame Functions.

Therefore, given the significance of the rate and order of convergence for iterative algorithms and the absence of such a result for Stochastic Subgradient Descent that can be applied to non-smooth machine learning models, we endeavour to produce experimental evidence that not only supports the existing theoretical results but more importantly will provide guidance for future theoretical research into the matter.

4.2.1 Rate and Order of Convergence Methodology

Our current aim is to provide experimental evidence for Stochastic Subgradient Descent's rate and order of convergence on a wide class of functions. We can achieve this by directly measuring the rate and order of convergence for various sequences of iterates produced by a run of Stochastic Subgradient Descent. Fortunately, this measurement can be done very simply as we now show.

Given a sequence of converging iterates $\{\mathbf{x_k}\}_{k\geq 0} \to \mathbf{x}^\star$, Equation (4.2.1) shows that:

$$\frac{e_{k+1}}{{e_k}^m} = \frac{\|\mathbf{x_{k+1}} - \mathbf{x}^\star\|}{\|\mathbf{x_k} - \mathbf{x}^\star\|^m} \approx \mu \tag{4.2.2}$$

with the approximation improving as $k \to \infty$. Taking the a logarithm of both sides and rearranging, we get:

$$\log(e_{k+1}) \approx m \log(e_k) + \log(\mu)$$

This relation implies that the line of best fit through the data $\{\log(e_k), \log(e_{k+1})\}_{k\geq 0}$ will have gradient m and intercept $\log(\mu)$. We can compute that data and its line of best fit, and so we can measure the rate and order of convergence given a sequence of iterates.

We note that our measurement of the order of convergence will be more accurate than the rate of convergence for this method: when completing linear regression and lines of best fit, the error for the intercept is almost always more significant than the error for the slope [55]. This issue is also exacerbated by the logarithms, as the error will grow exponentially as we move from $\log(\mu)$ to μ. Unfortunately, we can not overcome this issue whilst simultaneously measuring the rate and order of convergence. However, the order of the convergence is the most significant measurement and provides insight into the rate of convergence in the sublinear setting, i.e. $m < 1$ implies sublinear convergence, and so the issue's impact is somewhat mitigated.

With this measurement method established, we can complete our methodology by selecting which iterate sequences to measure. We need to ensure that these sequences are produced by applying Stochastic Subgradient Descent to functions that represent the class of functions satisfying the Iterate Assumptions for Stochastic Subgradient Descent and Lyapunov Assumptions for Stochastic Subgradient Descent. We selected such a group for our verification experiments, so we can use those functions again. Furthermore, as the measurement method only relies upon the final sequence of iterates, we can measure the rate and order of convergence for the same runs as before, streamlining our experimentation.

Overall, our methodology is to compute the line of best fit through the data $\{\log(e_k), \log(e_{k+1})\}_{k\geq 0}$ from our previous verification runs and use that line estimate Stochastic Subgradient Descent's rate and order of convergence.

4.2.2 Rate and Order of Convergence Results

We now analyse the results from our rate and order of convergence experiments. As we have used the same 110 total runs as our previous verification experiments, we do not report and analyse each run individually, but instead, summarise the results. A complete record of the experiments and the results can be found in Appendix Experimental Results.

Figure Order of Convergence Results summarises the order of convergence results:

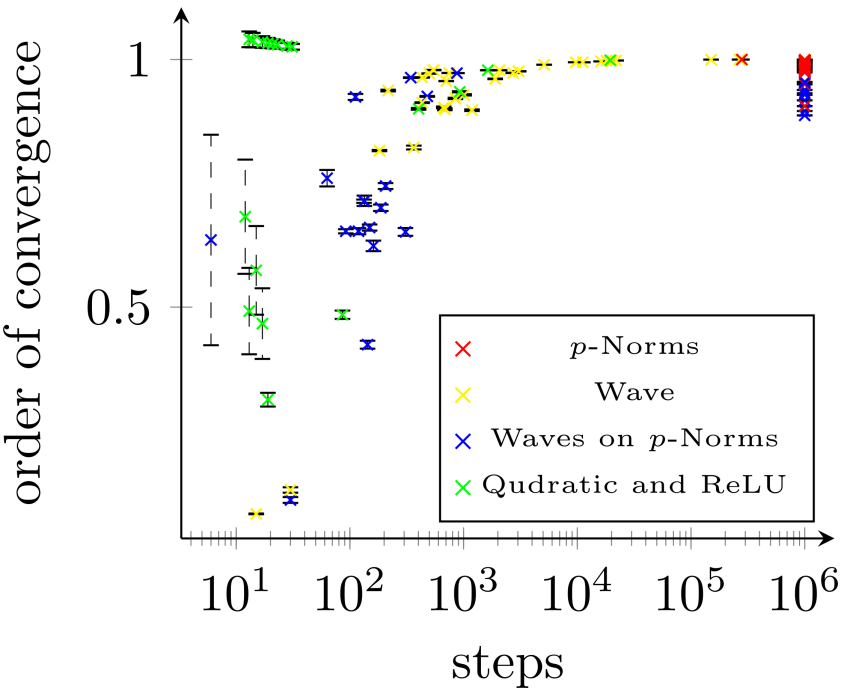


Figure 4.5: Order of Convergence Results

Immediately, we can identify a relation between step count and the reported value: as the number of steps increases, the order of convergence tends towards 1. Intuitively, this seems entirely wrong: the order of convergence should improve for runs that converge faster. However, despite this intuition, we can explain the phenomena. Our measurement method relies upon Equation (4.2.2) being a valid approximation, which is only true as the number of steps tends toward infinity. Therefore, we would expect the measured order of convergence would converge to its true value as the number of runs is increased. Given this trend, we can summarise the results by calculating a weighted average of the orders of convergence, where the weights are given by the number of steps in that run. If we do this, then we get a summary value of 0.976964. Furthermore, the error, measured by the variance, of these results is small and follows the same trend as the values themselves: the error tends towards 0 as the number of steps increases. As computing the line of best fit merely scales the error of the data points [55], this error is mostly representative of the noise of the iterates and so this trend is expected. Applying the same weighting method, we get a summary variance of 0.007561. This is a minimal relative error of 8% and so supports that our summary value is a good approximation of Stochastic Subgradient Descent's order of convergence. As this value is marginally less than 1, it suggests that Stochastic Subgradient Descent exhibits Q-sublinear convergence. This agrees with the current theoretical results for smooth and weakly convex functions, as outlined earlier [71, 45, 35]. Furthermore, given our varied choice of objective functions, these results support the hypothesis that Stochastic

Subgradient Descent exhibits sublinear convergence on a wider class of functions, potentially on the entire class of functions that satisfy the Iterate Assumptions for Stochastic Subgradient Descent and Lyapunov Assumptions for Stochastic Subgradient Descent.

Unfortunately, while the rate of convergence values do support the same hypothesis, they are not as conclusive. Figure Rate of Convergence Results summarises those results:

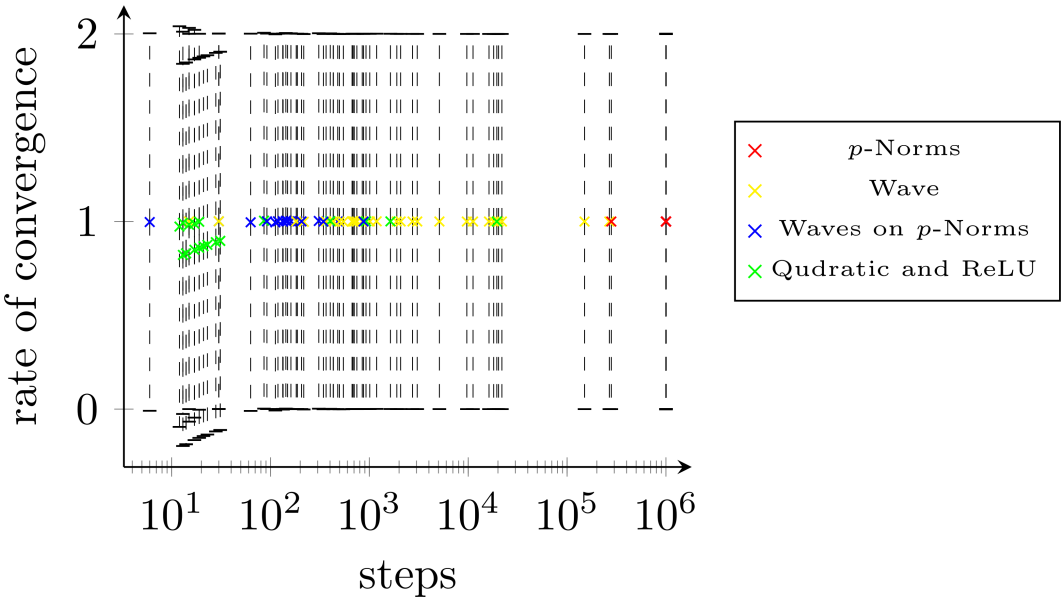


Figure 4.6: Rate of Convergence Results

Immediately, we can determine that the rate of convergence values are more consistent than the order of convergence values. They are clustered around 1 for all functions, except ReLU for which it sits around 0.85. Furthermore, they vary by at most 8% among runs of the same function and by 18% between functions. Thus, at first glance, the results would seem to provide strong experimental evidence that Stochastic Subgradient Descent's rate of convergence is sublinear, further supporting our hypothesis. Furthermore, we can identify the same relationship between the number of steps and the reported rate of convergence: as the number of steps increases, the reported value tends towards 1. If we employ the same weighting method as we did with the order of convergence, we get a summary value of 1.000028, providing further evidence for our hypothesis. However, there is a significant complicating factor. The weighted average of the variance is 1.003021: our measurement has a 100% relative error. This error is mostly representative of the uncertainty within the line of best fit, as mentioned when we outlined the measurement method. Due to this, the error only lightly follows the same trend as the values themselves, moving slightly closer to 1 as the number of steps increases. This does not resolve the issue but rather worsens it, providing further evidence of the large relative error. Frustratingly, as we also mentioned earlier, this error is

unavoidable and our particular setting just worsens its impact. As a result, we can only manage this error rather than lessen it. Thus, we are forced to accept the rate of convergence evidence with considerable caution. The reported value still has significance: it is optimal for the given data and so is least likely to be the wrong value within the given bound. However, further experimentation could still prove that this value is incorrect and our errors suggest that the true value could be significantly different. Therefore, while the rate of convergence results also suggest that Stochastic Subgradient Descent converges sublinearly on a wide class of functions, they are not very certain.

Overall, both the order and rate of convergence results provide reasonable evidence for the hypothesis that Stochastic Subgradient Descent converges Q-sublinearly on a wide class of functions, potentially even the entire class of functions that satisfy the Iterate Assumptions for Stochastic Subgradient Descent and Lyapunov Assumptions for Stochastic Subgradient Descent. Given that only the order of convergence results are very certain, further experimentation is needed to strengthen the rate of convergence results and decisively evidence the hypothesis.

4.2.3 Conclusion of Rate and Order of Convergence Experiments

In this section, we examined the speed of Stochastic Subgradient Descent to assess its practical viability to train non-smooth machine learning models. We began by outlining exactly why the speed of an algorithm is of critical importance and then introduced Q Convergence as the rigorous definition of speed for an iterative algorithm. However, despite being a crucial result for practical application, a theoretical result regarding Stochastic Subgradient Descent's Q convergence on the class of functions that satisfy the Iterate Assumptions for Stochastic Subgradient Descent and Lyapunov Assumptions for Stochastic Subgradient Descent is yet to be proven. Existing theory only shows that Stochastic Subgradient Descent achieves sublinear convergence in smooth and weakly convex settings. Given this, we aimed to produce experimental evidence that supported the limited existing theory and also provide guidance for further research. Using a measurement method that estimates the rate and order of convergence by computing a line of best fit on the data $\{(e_{k+1}, e_k)\}_{k\geq 0}$, we achieved this goal: our results support the existing theory and provide substantial enough evidence to conjecture that Stochastic Subgradient Descent's sublinear convergence extends to a wider class of functions, possibly the entire class of functions that satisfy the Iterate Assumptions for Stochastic Subgradient Descent and Lyapunov Assumptions for Stochastic Subgradient Descent.

4.3 Extending Theory and Application

We now review our experimental work and what the results imply regarding the application of Stochastic Subgradient Descent, in particular to the training of non-smooth machine learning models. After this assessment, we then outline future research directions that would build upon the theory and application of the algorithm and potentially

lead to improvements in its practical use.

4.3.1 Summary of Results

Our experimental work led to three different results. Firstly, we successfully implemented the first generic version of Stochastic Subgradient Descent: it can be applied to any optimisation problem of the form (2.1.2) and its hyperparameters, such as the sequence of step sizes and sampling method, are entirely determined by the user. Overcoming computational difficulties and restrictions, this implementation verified the translation of the theoretical algorithm to a practical real-world setting.

Secondly, we tested this implementation on a group of functions representative of those that satisfy the Iterate Assumptions for Stochastic Subgradient Descent and Lyapunov Assumptions for Stochastic Subgradient Descent. Many of these tests converged within our set 1 million step limit and those that did not still exhibited behaviour that strongly suggests they were converging. Thus, these tests provided significant evidence that the Convergence of Stochastic Subgradient Descent Under Practical Assumptions translates to real-world application.

Finally, we measured the rate and order of convergence for our tests. Factoring in the approximation error related to the number of steps in each run, we found the order of convergence to be 0.976964 ± 0.007561. Unfortunately, a modelling error led to a substantially less confident result for the rate of convergence: 1.000028 ± 1.003021. That said, with these mixed outcomes, we were still able to verify existing results which show that Stochastic Subgradient Descent converges Q-sublinearly on smooth and weakly convex objective functions, and provide reasonable evidence that this convergence extends to the functions that satisfy the Iterate Assumptions for Stochastic Subgradient Descent and Lyapunov Assumptions for Stochastic Subgradient Descent.

In total, these three experimental results all validate the practical application of Stochastic Subgradient Descent for the training of non-smooth machine learning models: the algorithm can be implemented and applied; it will converge and so effectively train the model; and finally, its runtime is very competitive in comparison to other optimisation methods.

4.3.2 Future Research

Given the current state of the theory and application of Stochastic Subgradient Descent, there are various directions that future research could go. We outline a few such directions that we deem particularly interesting and promising.

Firstly, and most obviously, an extension of the existing theory for Stochastic Subgradient Descent's rate and order of convergence to Tame Functions, or even the entire class of functions that satisfy the Iterate Assumptions for Stochastic Subgradient Descent and Lyapunov Assumptions for Stochastic Subgradient Descent, would be a very useful

result. Unfortunately, we suspect that such an extension would not arise easily from the existing proofs, as they rely upon convergence theory simpler than that required to prove the Convergence of Stochastic Subgradient Descent Under Practical Assumptions. That said, hopefully, experimental data and results such as our own will provide some guidance for the theorem statement and so remove some difficulty.

Another promising and interesting research direction would be to examine, develop theory, and verify the application of extensions and variations of Stochastic Subgradient Descent. There are many paths that this research could take. Extensions that expand the domain of application would be immensely helpful for applying Stochastic Subgradient Descent in wider settings. Some such extensions already exist: a proximal extension of Stochastic Subgradient Descent to constrained optimisation problems has been defined and proven to converge under similar conditions to the base algorithm [36]. On the other hand, variations may be able to improve the performance, be that in the optimal value, the rate and order of convergence, or another metric. As previously noted, variants employing linesearches and accelerated techniques have already been developed and assessed for their improvements [106, 107]. Given the similarities between Stochastic Gradient Descent and Stochastic Subgradient Descent, many existing extensions and variants of the former will port across naturally to the latter, easing the process and providing a good foundation for theoretical analysis and application. For example, a Stochastic Subgradient Descent that incorporates a momentum term will probably arise naturally from one of the Stochastic Gradient Descent momentum-based variants [78].

Unfortunately, higher-order methods, such as Newton's Method or Adam [57], can not be ported across as there is no generalisation of the Clarke Subdifferential to higher derivatives. However, trying to construct such a generalisation may be possible and so itself worth future research. This would certainly move into the realm of variational analysis, building upon the theory of Rockafellar [82], but a higher-order version of Stochastic Subgradient Descent could exhibit significantly better performance and so provides ample motivation for the development of such theory.

Even within vanilla Stochastic Subgradient Descent, there is capacity for improvements: the sampling process for $\mathbf{g_k} \in \partial f(\mathbf{x_k})$ is left entirely open. The Convergence of Stochastic Subgradient Descent Under Practical Assumptions assume random sampling, so we can infer that any choice of $\mathbf{g_k}$ will still result in convergence. However, particular choices, such as the 'steepest' value, may result in improved rates and order of convergence. This must be balanced with the computational time required to make an informed choice, but these heuristic methods have exhibited reasonable performance in other optimisation algorithms [111]. We would suspect that a No Free Lunch Theorem would apply, preventing these methods from being universally superior, but these results only apply to all mathematically possible problems, rather than just those that occur within real-world limitations, and so the heuristic methods may still provide practical improvements.

On the application side, further research into approximation and computation of the

Clarke Subdifferential would be very helpful. The computational requirements for the Clarke Subdifferential far outweigh that of a simple gradient. This is most clearly evidenced by the sampling approximation method we employed which calculates many gradients within a local neighbourhood [26]. As a result, the total computational steps for an implementation of Stochastic Subgradient Descent will most likely be significantly greater than that of a similar implementation of Stochastic Gradient Descent. So while the order and rate of convergence of the two algorithms are near-identical, the computational requirements related to the Clarke Subdifferential may limit the former's application. This is a major hurdle and so further research into computationally efficient approximations and computations of the Clarke Subdifferential would be very beneficial.

Finally, further investigation into the relationship between a Stochastic Subgradient Descent variant using the sampling approximation method and the stochastic gradient sampling descent method [58, 25] could yield various insightful and practical results. As noted when we introduced the sampling method - see (3.3.1) - the similarities between the two allow for analysis to cross with relative ease. Given that the stochastic gradient sampling descent method is a well-established variant of Stochastic Gradient Descent, its theoretical foundation is much more robust. Therefore, there may be results that can cross to Stochastic Subgradient Descent and address some of the above issues and problems.

Chapter 5

Conclusion

As stated in the Introduction, the principal goal of this thesis was to examine Stochastic Subgradient Descent and determine its applicability to training non-smooth machine learning problems.

We began this task by formally outlining the Background required to understand and analyse the results required for such examination. This involved the following three sections: an Introduction to Optimisation, in which we set out the taxonomy and tools that enable us to precisely specify the optimisation setting to which Stochastic Subgradient Descent applies, i.e. global, unconstrained, continuous, non-smooth, and stochastic problems; a consideration of Smooth Optimisation, deriving the First-Order Optimality Condition for Smooth Optimisation which we then used to prove Convergence of Stochastic Gradient Descent; and finally, a summary of Machine Learning that served to both motivate our interest and show that many machine learning models are specific instances of non-smooth optimisation problems.

With this background complete, we could perform our examination Stochastic Subgradient Descent, beginning with its theory. We opened this chapter by assessing the viability of Applying Smooth Optimisation Techniques to Machine Learning and determined that smooth algorithms, such as Stochastic Gradient Descent, can not be used due to unsolvable problems related to the non-smooth structure of machine learning methods. As a result, we turned to Non-Smooth Optimisation, introducing the Clarke Subdifferential and its various properties and relations, and showed how Stochastic Subgradient Descent is developed by Generalising Stochastic Gradient Descent. However, while Stochastic Subgradient Descent overcomes the issues of smooth optimisation techniques and can be applied to non-smooth machine learning problems, the convergence of Stochastic Gradient Descent does not generalise and so must be proven separately. Given that a guarantee of convergence is a critical requirement for practical application, we then sought to prove this Convergence. Following recent research developments [36], we ex-

amined the Convergence of General Dynamic Systems using differential inclusions and then applied the results to determine the Convergence of Subdifferential Dynamic Systems. This enabled us to prove that two sets of assumptions, the Iterate Assumptions for Stochastic Subgradient Descent and Lyapunov Assumptions for Stochastic Subgradient Descent, are satisfactory for guaranteeing the Convergence of Stochastic Subgradient Descent Under Practical Assumptions. To make the link between the assumptions and non-smooth machine learning models explicit, we then investigated Sufficient Classes of Functions, showing that the Tame Functions are a wide class of functions that satisfy the required assumptions for convergence. Thus, after a Discussion of the Theory in which we resolved various technicalities, we proved that the Application of Stochastic Subgradient Descent to Supervised Learning Models is backed by rigorous convergence theory.

We then turned to experimental work to verify the Application of Theory. We had two goals: Verifying Convergence in a real-world setting and determining the Rate and Order of Convergence for Stochastic Subgradient Descent, as it is an important factor in its viability as a practical method. To achieve the first goal, we had to produce the first generic Implementation of Stochastic Subgradient Descent, for which we needed to overcome a handful of computational and approximation issues. With this implementation, we set out a Verification Methodology that enabled us to test the convergence of the algorithm on a group of functions representative of the Tame Functions. The Verification Results produced through this implementation and methodology provided significant experimental evidence that the Convergence of Stochastic Subgradient Descent Under Practical Assumptions translates to real-world applications. For the second goal, we formally defined the rate and order of convergence of an iterative algorithm through Q Convergence and illustrated that the current theory only proved that Stochastic Subgradient Descent achieved Q-sublinear convergence in very limited settings. Given this limited foundation, we developed a methodology to verify these results and test the hypothesis that Stochastic Subgradient Descent also achieves Q-sublinear convergence on the entire class of functions that satisfy the assumptions for convergence. Our Rate and Order of Convergence Results did verify the existing theory and provided reasonable experimental evidence supporting the hypothesis, however, issues relating to the error of the rate of convergence prevent any definite conclusions. A Summary of Results allowed us to conclude that Stochastic Subgradient Descent can be applied to the training of non-smooth machine learning methods in a practical, real-world setting. Finally, with this basis, we identified and examined Future Research directions that could further extend Stochastic Subgradient Descent and improve its application.

Overall, we have achieved our set goal: we examined Stochastic Subgradient Descent in-depth, from both a theoretical and application standpoint, and determined that it can be applied successfully to the training of machine learning models.

Appendix A

Experimental Results

Quadratic Function Objective function is

$$f(x) = ax^2 + bx + c$$

with $a \in (0, 1)$, $b \in (-10, 10)$, and $c \in (-10, 10)$:

Table A.1: Quadratic Function - Verification Experiments

run	a	b	c	initial iterate	steps	final iterate
0	0.822762	-8.686135	7.794542	[9.10519]	15	[5.27865]
1	0.212855	8.277879	-5.060770	[5.085218]	402	[-19.444817]
2	0.373557	-2.164032	5.680585	[7.466167]	86	[2.896534]
3	0.960411	-8.342047	0.669460	[7.019532]	12	[4.342957]
4	0.114639	9.771320	-6.170564	[9.87441]	1634	[-42.617579]
5	0.036515	-9.732315	-5.724879	[9.13237]	19476	[133.263942]
6	0.783826	-9.676055	8.407215	[7.046427]	13	[6.172324]
7	0.632307	-8.473146	1.828305	[7.522743]	19	[6.700172]
8	0.122479	-2.758167	-2.706938	[7.109791]	925	[11.259674]
9	0.711004	-8.340570	-9.169306	[9.725367]	17	[5.865345]

Table A.2: Quadratic Function - Rate and Order of Convergence Experiments

run	order	variance of order	rate	variance of rate
0	0.574247	0.089524	0.982801	1.049629
1	0.900410	0.001808	1.002324	1.000680
2	0.484657	0.008471	1.004508	1.001818
3	0.682616	0.115355	0.973112	1.067884
4	0.978448	0.000304	1.000759	1.000147
5	0.998572	0.000010	1.000081	1.000006
6	0.492025	0.087259	0.993091	1.018997
7	0.313100	0.013874	0.998837	1.002070
8	0.934934	0.001107	1.000941	1.000171
9	0.466739	0.071343	0.988248	1.033615

ReLU Objective function is

$$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$$

Table A.3: ReLU Function - Verification Experiments

run	initial iterate	steps	final iterate
0	[5.161614]	13	[-0.08476]
1	[6.166946]	17	[-0.000757]
2	[8.643201]	31	[-0.023002]
3	[8.167266]	28	[-0.028014]
4	[8.597666]	31	[-0.068536]
5	[6.420183]	19	[-0.162007]
6	[7.161394]	23	[-0.18138]
7	[5.342969]	14	[-0.147372]
8	[5.360852]	14	[-0.129489]
9	[6.88266]	21	[-0.09004]

Table A.4: ReLU Function - Rate and Order of Convergence Experiments

run	order	variance of order	rate	variance of rate
0	1.040903	0.015905	0.822075	1.018651
1	1.035458	0.011700	0.849577	1.014583
2	1.025780	0.005793	0.896821	1.008235
3	1.027229	0.006535	0.890003	1.009090
4	1.025780	0.005793	0.896821	1.008235
5	1.033439	0.010288	0.859703	1.013147
6	1.030232	0.008236	0.875543	1.010977
7	1.039307	0.014617	0.830136	1.017435
8	1.039307	0.014617	0.830136	1.017435
9	1.031720	0.009159	0.868235	1.011966

Scaled p-Norms Objective function is

$$f(x) = a\|\mathbf{x}\|_p = a\left(\sum_{i=1}^{d}|x_i|^p\right)^{\frac{1}{p}}$$

with $p \in \{1, 2, \infty\}$ and $a \in (0, 1)$:

Table A.5: Scaled 1-Norm - Verification Experiments

run	a	initial iterate	steps	final iterate
0	0.435486	[5.809185 9.595118]	1000000	[-1.3e-05 0.0e+00]
1	0.551466	[7.157019 5.995037]	1000000	[2.0e-04 9.5e-05]
2	0.273481	[8.388968 5.858762]	1000000	[-0.000119 -0.]
3	0.109748	[6.609327 8.160881]	1000000	[0.0e+00 2.6e-05]
4	0.796386	[5.149327 5.728466]	1000000	[-8.40e-05 -3.99e-04]
5	0.014550	[7.357077 5.227534]	280164	[1.e-06 8.e-06]
6	0.554961	[8.897593 5.52669]	1000000	[1.34e-04 -1.60e-05]
7	0.211879	[7.25798 7.752547]	1000000	[-1.40e-05 -1.06e-04]
8	0.523753	[8.640273 8.989127]	1000000	[1.14e-04 -9.80e-05]
9	0.393165	[8.53227 7.611877]	1000000	[0.0e+00 -1.5e-05]

Table A.6: Scaled 1-Norm - Rate and Order of Convergence Experiments

run	order	variance of order	rate	variance of rate
0	0.991814	0.000021	1.000002	1.000000
1	0.973921	0.000080	1.000011	1.000001
2	0.995770	0.000006	1.000001	1.000000
3	0.999358	0.000002	1.000000	1.000000
4	0.905949	0.000241	1.000070	1.000002
5	0.999992	0.000000	0.999997	1.000000
6	0.984664	0.000021	1.000005	1.000000
7	0.997275	0.000007	1.000000	1.000000
8	0.987334	0.000041	1.000005	1.000001
9	0.991963	0.000014	1.000002	1.000000

Table A.7: Scaled 2-Norm - Verification Experiments

run	a	initial iterate	steps	final iterate
0	0.786028	[9.163715 7.957487]	1000000	[0.000125 -0.000228]
1	0.258860	[9.744635 6.426404]	1000000	[7.0e-05 2.1e-05]
2	0.753811	[8.138968 6.573113]	1000000	[6.90e-05 1.51e-04]
3	0.680795	[6.708441 6.86071]	1000000	[-2.5e-05 -7.0e-06]
4	0.842450	[8.739444 5.795273]	1000000	[1.16e-04 -3.70e-05]
5	0.629133	[7.808576 5.871137]	1000000	[0.0e+00 -2.3e-05]
6	0.977417	[7.218955 5.286167]	1000000	[-2.4e-05 9.7e-05]
7	0.181148	[7.066561 9.635923]	1000000	[1.3e-05 6.2e-05]
8	0.172621	[6.177335 5.507269]	1000000	[4.0e-05 -1.2e-05]
9	0.385328	[5.678233 9.232774]	1000000	[1.68e-04 1.00e-05]

Table A.8: Scaled 2-Norm - Rate and Order of Convergence Experiments

run	order	variance of order	rate	variance of rate
0	0.983457	0.000070	1.000007	1.000001
1	0.998374	0.000005	1.000000	1.000000
2	0.979073	0.000079	1.000009	1.000001
3	0.980807	0.000056	1.000007	1.000001
4	0.975430	0.000074	1.000011	1.000001
5	0.983708	0.000058	1.000005	1.000001
6	0.949835	0.000147	1.000027	1.000001
7	0.999283	0.000002	0.999999	1.000000
8	0.998460	0.000005	1.000000	1.000000
9	0.995412	0.000021	1.000001	1.000001

Table A.9: Scaled ∞-Norm - Verification Experiments

run	a	initial iterate	steps	final iterate
0	0.581680	[5.625721 5.098367]	1000000	[2.25e-04 -3.00e-06]
1	0.646206	[6.447542 7.002138]	1000000	[2.52e-04 7.20e-05]
2	0.168818	[9.092081 5.569706]	1000000	[-2.8e-05 -1.5e-05]
3	0.923761	[9.548746 5.990823]	1000000	[-3.3e-05 -4.3e-04]
4	0.519234	[7.006274 5.642572]	1000000	[-7.9e-05 7.2e-05]
5	0.166514	[8.797795 5.714182]	1000000	[-6.0e-06 -7.7e-05]
6	0.441310	[7.518667 8.576632]	1000000	[3.6e-05 -3.6e-05]
7	0.895153	[7.065971 5.501172]	1000000	[-0.000161 0.000161]
8	0.163573	[5.570195 9.469185]	1000000	[5.8e-05 2.4e-05]
9	0.329315	[8.176206 7.006257]	1000000	[5.6e-05 -5.6e-05]

Table A.10: Scaled ∞-Norm - Rate and Order of Convergence Experiments

run	order	variance of order	rate	variance of rate
0	0.987331	0.000053	1.000005	1.000001
1	0.991343	0.000026	1.000004	1.000000
2	0.999582	0.000002	1.000000	1.000000
3	0.985856	0.000059	1.000010	1.000001
4	0.993493	0.000023	1.000002	1.000000
5	0.999588	0.000001	1.000000	1.000000
6	0.997363	0.000012	1.000000	1.000000
7	0.977758	0.000091	1.000013	1.000001
8	0.999632	0.000001	1.000000	1.000000
9	0.998349	0.000008	1.000000	1.000000

p-Norm Waves Objective function is

$$f(x) = a\cos\left(\frac{\|\mathbf{x}\|_p}{b}\right)$$

with $p \in \{1, 2, \infty\}$, $a \in (1, 2)$, and $b \in (1, 2)$:

Table A.11: 1-Norm Wave - Verification Experiments

run	a	b	initial iterate	steps	final iterate
0	1.614308	1.619428	[7.194074 6.408116]	9690	[10.529843 9.743886]
1	1.345262	1.211539	[7.197088 5.587364]	847	[6.150964 4.54124]
2	1.302791	1.446334	[8.114969 8.475737]	498	[11.795015 12.155783]
3	1.165184	1.938804	[8.374403 7.293873]	16226	[10.593873 9.513343]
4	1.954966	1.674494	[7.236365 5.723685]	182	[4.92224 3.409561]
5	1.928377	1.824986	[8.613092 5.284197]	150159	[21.739117 18.410223]
6	1.921969	1.476371	[9.952111 9.986637]	30	[9.192043 9.226568]
7	1.440236	1.649918	[6.295173 9.239774]	432	[2.617566 5.562167]
8	1.132072	1.199389	[7.160342 5.3068]	432	[4.832087 2.978545]
9	1.779428	1.918445	[9.040697 9.330484]	2734	[8.299444 8.589232]

Table A.12: 1-Norm Wave - Rate and Order of Convergence Experiments

run	order	variance of order	rate	variance of rate
0	0.994470	0.000046	1.000112	1.000008
1	0.921490	0.001402	1.000674	1.000115
2	0.971799	0.000454	0.999528	1.000157
3	0.996338	0.000029	1.000067	1.000004
4	0.816158	0.001250	1.001122	1.000235
5	0.999830	0.000000	1.000006	1.000000
6	0.130704	0.005460	1.001480	1.000747
7	0.963845	0.000529	0.999573	1.000174
8	0.912972	0.000730	1.000754	1.000127
9	0.973104	0.000399	1.000224	1.000024

Table A.13: 2-Norm Wave - Verification Experiments

run	a	b	initial iterate	steps	final iterate
0	1.471364	1.976005	[5.045833 6.819806]	21914	[9.669026 13.068385]
1	1.080934	1.460603	[9.946171 7.556343]	748	[13.942022 10.592086]
2	1.888791	1.089622	[5.539587 6.669131]	1000000	[12.60413 15.174162]
3	1.976455	1.888310	[5.524631 8.623408]	669	[4.633902 7.233068]
4	1.763653	1.751483	[7.618888 6.75353]	1010	[7.002125 6.20682]
5	1.060675	1.614050	[5.487008 8.377035]	546	[2.81356 4.295472]
6	1.201165	1.562957	[5.814595 9.047696]	703	[3.379338 5.258357]
7	1.291993	1.955463	[6.44681 8.861063]	11183	[8.077666 11.102656]
8	1.436912	1.115215	[6.292004 8.664697]	1896	[5.917675 8.14921]
9	1.963864	1.721136	[7.606344 7.65846]	217	[4.006116 4.033564]

Table A.14: 2-Norm Wave - Rate and Order of Convergence Experiments

run	order	variance of order	rate	variance of rate
0	0.998094	0.000010	1.000050	1.000003
1	0.972941	0.000138	0.999983	1.000039
2	1.000002	0.000000	0.999995	1.000000
3	0.903438	0.001771	1.000849	1.000159
4	0.928738	0.001343	1.000453	1.000070
5	0.978807	0.000429	0.999417	1.000157
6	0.956507	0.000185	1.000362	1.000041
7	0.994434	0.000050	1.000091	1.000006
8	0.961014	0.000660	1.000231	1.000028
9	0.937391	0.001714	0.998771	1.000602

Table A.15: ∞-Norm Wave - Verification Experiments

run	a	b	initial iterate	steps	final iterate
0	1.721097	1.503784	[7.414143 9.347289]	1185	[7.261682 7.261692]
1	1.708571	1.999057	[8.080317 7.20059]	18092	[16.591585 7.20059]
2	1.927981	1.906961	[9.071919 6.973372]	267732	[32.175926 6.973372]
3	1.111585	1.254336	[8.398434 6.330369]	3050	[9.616107 6.330369]
4	1.776264	1.066286	[8.099621 9.411848]	15	[8.099621 10.979217]
5	1.790627	1.999326	[7.534295 8.126384]	5110	[7.534295 12.843927]
6	1.755897	1.767661	[6.967552 7.999681]	365	[6.967552 7.272874]
7	1.070246	1.591827	[7.848355 9.453889]	2072	[5.703661 5.703661]
8	1.853490	1.585838	[9.632982 7.855901]	684	[8.567435 7.855901]
9	1.579957	1.198123	[5.724376 9.803533]	20275	[5.724376 16.67929]

Table A.16: ∞-Norm Wave - Rate and Order of Convergence Experiments

run	order	variance of order	rate	variance of rate
0	0.897657	0.001634	1.001142	1.000125
1	0.997785	0.000012	1.000057	1.000003
2	0.999921	0.000000	1.000003	1.000000
3	0.976285	0.000335	1.000218	1.000023
4	0.082475	0.000800	1.000408	1.000202
5	0.989600	0.000102	1.000193	1.000018
6	0.822352	0.003866	1.000898	1.000199
7	0.978383	0.000232	1.000356	1.000042
8	0.899617	0.001958	1.000700	1.000127
9	0.997760	0.000014	1.000056	1.000003

p-Norms Subtracting p-Norm Waves Objective function is

$$f(x) = c\|\mathbf{x}\|_p - a\cos\left(\frac{\|\mathbf{x}\|_p}{b}\right)$$

with $p \in \{1, 2, \infty\}$, $a \in (1, 2)$, $b \in (1, 2)$, and $c \in (0, 2)$:

Table A.17: 1-Norm Subtracting 1-Norm Wave - Verification Experiments

run	a	b	c	initial iterate	steps	final iterate
0	1.751904	1.708644	0.630389	[9.319379 6.641391]	63	[6.141165 3.463177]
1	1.543735	1.258438	0.454185	[7.484584 7.779166]	6	[7.521059 7.815641]
2	1.767737	1.924906	0.405265	[9.507804 9.178159]	112	[5.772216 5.442572]
3	1.603760	1.587573	1.906986	[9.18415 8.273173]	1000000	[-0.000956 -0.]
4	1.358406	1.631592	1.602135	[8.453532 7.917651]	1000000	[-0. 0.000803]
5	1.035955	1.344762	1.988247	[6.839933 5.188796]	1000000	[-0.000997 -0.]
6	1.635627	1.377095	1.242474	[9.937379 5.657128]	1000000	[0.000623 -0.]
7	1.579773	1.448684	1.154980	[9.906964 5.770951]	1000000	[-0. -0.000579]
8	1.093250	1.593206	1.496251	[7.799218 5.3536]	1000000	[0. 0.00075]
9	1.521682	1.716117	1.878553	[9.065209 9.720598]	1000000	[0.000942 -0.]

Table A.18: 1-Norm Subtracting 1-Norm Wave - Rate and Order of Convergence Experiments

run	order	variance of order	rate	variance of rate
0	0.760411	0.016639	0.995938	1.005627
1	0.635619	0.212575	0.997260	1.006197
2	0.924553	0.006028	0.995469	1.002736
3	0.928839	0.000145	1.000159	1.000001
4	0.917666	0.000142	1.000143	1.000001
5	0.887272	0.000221	1.000263	1.000001
6	0.995045	0.000027	1.000006	1.000001
7	0.991923	0.000033	1.000010	1.000001
8	0.896050	0.000173	1.000181	1.000001
9	0.931916	0.000137	1.000149	1.000001

Table A.19: 2-Norm Subtracting 2-Norm Wave - Verification Experiments

run	a	b	c	initial iterate	steps	final iterate
0	1.899691	1.975847	0.690541	[7.527543 5.156397]	479	[8.935965 6.121172]
1	1.909610	1.413046	0.003971	[5.636209 7.775241]	30	[5.208408 7.185083]
2	1.166359	1.604310	0.047678	[5.605409 7.444503]	187	[6.000003 7.968561]
3	1.046039	1.743771	0.700976	[8.517147 5.870555]	1000000	[0.000145 0.0001]
4	1.215552	1.997571	1.177677	[6.588429 5.133313]	1000000	[0.000233 0.000181]
5	1.811594	1.259149	1.277805	[6.265995 7.835885]	143	[4.081231 5.103748]
6	1.372116	1.050012	1.195203	[6.241255 6.977088]	161	[3.590313 4.013604]
7	1.531288	1.980837	0.828318	[8.046578 7.007952]	1000000	[0.000157 0.000136]
8	1.983948	1.688713	0.998721	[7.468514 8.463559]	308	[5.885087 6.66917]
9	1.587473	1.704611	0.100633	[7.980606 8.937189]	149	[7.010854 7.8512]

Table A.20: 2-Norm Subtracting 2-Norm Wave - Rate and Order of Convergence Experiments

run	order	variance of order	rate	variance of rate
0	0.925566	0.000804	1.000529	1.000098
1	0.110524	0.005828	1.001227	1.000597
2	0.700687	0.006494	1.001242	1.000342
3	0.997463	0.000012	1.000000	1.000000
4	0.938697	0.000103	1.000039	1.000001
5	0.424181	0.007965	1.003876	1.001120
6	0.623698	0.010645	1.001573	1.001757
7	0.998755	0.000009	1.000000	1.000000
8	0.651897	0.007872	1.002999	1.000742
9	0.660758	0.006187	1.002023	1.000616

Table A.21: ∞-Norm Subtracting ∞-Norm Wave - Verification Experiments

run	a	b	c	initial iterate	steps	final iterate
0	1.924797	1.060577	1.374396	[8.679275 9.309925]	134	[5.752586 5.752595]
1	1.708043	1.671463	0.129383	[5.350027 8.316571]	133	[5.350027 10.2899]
2	1.469216	1.242154	1.833486	[5.361988 8.970841]	1000000	[1.7e-05 -4.6e-04]
3	1.487735	1.428746	0.633620	[7.37176 6.259918]	119	[8.042421 6.259918]
4	1.487439	1.528832	1.087211	[9.739787 9.228024]	1000000	[0.000272 0.000116]
5	1.445676	1.368189	1.774932	[7.263404 5.201324]	1000000	[-0.000253 -0.000445]
6	1.187359	1.965590	0.245883	[7.832577 7.956949]	873	[7.832577 11.526131]
7	1.495704	1.671230	0.592375	[7.769354 8.981828]	206	[7.769354 9.291754]
8	1.744591	1.685539	0.638030	[6.666929 5.411134]	343	[9.470988 5.411134]
9	1.139594	1.184790	0.425962	[9.707255 6.305679]	92	[6.900711 6.305679]

Table A.22: ∞-Norm Subtracting ∞-Norm Wave - Rate and Order of Convergence Experiments

run	order	variance of order	rate	variance of rate
0	0.714779	0.010495	0.999800	1.002278
1	0.713940	0.003370	1.001649	1.000459
2	0.954873	0.000124	1.000046	1.000001
3	0.653361	0.006164	1.001251	1.000389
4	0.998636	0.000011	1.000000	1.000001
5	0.952948	0.000119	1.000047	1.000001
6	0.972557	0.000073	1.000146	1.000016
7	0.744619	0.005976	1.000683	1.000172
8	0.963445	0.000635	0.999477	1.000170
9	0.653209	0.004197	1.001077	1.000779

Bibliography

[1] 2019. Ieee standard for floating-point arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, (2019), 1–84. doi:10.1109/IEEESTD.2019.8766229. [Cited on pages 69 and 85.]

[2] Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al., 2016. {TensorFlow}: A system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 265–283. [Cited on pages 36 and 75.]

[3] Abiodun, O. I.; Jantan, A.; Omolara, A. E.; Dada, K. V.; Mohamed, N. A.; and Arshad, H., 2018. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4, 11 (2018), e00938. [Cited on page 31.]

[4] Aizerman, M. A., 1964. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and remote control*, 25 (1964), 821–837. [Cited on page 29.]

[5] Al-Sahaf, H.; Bi, Y.; Chen, Q.; Lensen, A.; Mei, Y.; Sun, Y.; Tran, B.; Xue, B.; and Zhang, M., 2019. A survey on evolutionary machine learning. *Journal of the Royal Society of New Zealand*, 49, 2 (2019), 205–228. [Cited on page 4.]

[6] Alexopoulos, C.; Lachana, Z.; Androutsopoulou, A.; Diamantopoulou, V.; Charalabidis, Y.; and Loutsaris, M. A., 2019. How machine learning is changing e-government. In *Proceedings of the 12th International Conference on Theory and Practice of Electronic Governance*, 354–363. [Cited on page 1.]

[7] Amazon, 2022. Machine learning and artificial intelligence - amazon web services. https://aws.amazon.com/machine-learning/. [Cited on page 23.]

[8] Apicella, A.; Donnarumma, F.; Isgrò, F.; and Prevete, R., 2021. A survey on modern trainable activation functions. *Neural Networks*, 138 (2021), 14–32. [Cited on page 76.]

[9] Audet, C. and Hare, W., 2017. *Derivative-free and blackbox optimization*, vol. 2. Springer. [Cited on pages 11, 42, and 43.]

[10] Avis, D.; Bremner, D.; and Seidel, R., 1997. How good are convex hull algorithms? *Computational Geometry*, 7, 5-6 (1997), 265–301. [Cited on page 81.]

[11] Avron, H.; Kale, S.; Kasiviswanathan, S.; and Sindhwani, V., 2012. Efficient and practical stochastic subgradient descent for nuclear norm regularization. *arXiv preprint arXiv:1206.6384*, (2012). [Cited on pages 3 and 80.]

[12] Badue, C.; Guidolini, R.; Carneiro, R. V.; Azevedo, P.; Cardoso, V. B.; Forechi, A.; Jesus, L.; Berriel, R.; Paixao, T. M.; Mutz, F.; et al., 2021. Self-driving cars: A survey. *Expert Systems with Applications*, 165 (2021), 113816. [Cited on page 23.]

[13] Barber, C. B.; Dobkin, D. P.; and Huhdanpaa, H., 1996. The quickhull algorithm for convex hulls. *ACM TRANSACTIONS ON MATHEMATICAL SOFTWARE*, 22, 4 (1996), 469–483. [Cited on page 81.]

[14] Baydin, A. G.; Pearlmutter, B. A.; Radul, A. A.; and Siskind, J. M., 2018. Automatic differentiation in machine learning: a survey. *Journal of Marchine Learning Research*, 18 (2018), 1–43. [Cited on page 74.]

[15] Bianchi, P.; Hachem, W.; and Schechtman, S., 2022. Convergence of constant step stochastic gradient descent for non-smooth non-convex functions. *Set-Valued and Variational Analysis*, (2022), 1–31. [Cited on page 4.]

[16] Blondel, M.; Berthet, Q.; Cuturi, M.; Frostig, R.; Hoyer, S.; Llinares-López, F.; Pedregosa, F.; and Vert, J.-P., 2021. Efficient and modular implicit differentiation. *arXiv preprint arXiv:2105.15183*, (2021). [Cited on page 32.]

[17] Bolte, J.; Daniilidis, A.; Lewis, A.; and Shiota, M., 2007. Clarke subgradients of stratifiable functions. *SIAM Journal on Optimization*, 18, 2 (2007), 556–572. [Cited on page 62.]

[18] Bolte, J.; Le, T.; Pauwels, E.; and Silveti-Falls, T., 2021. Nonsmooth implicit differentiation for machine-learning and optimization. *Advances in Neural Information Processing Systems*, 34 (2021). [Cited on page 4.]

[19] Bolte, J. and Pauwels, E., 2020. A mathematical model for automatic differentiation in machine learning. *Advances in Neural Information Processing Systems*, 33 (2020), 10809–10819. [Cited on page 74.]

[20] Bolte, J. and Pauwels, E., 2021. Conservative set valued fields, automatic differentiation, stochastic gradient methods and deep learning. *Mathematical Programming*, 188, 1 (2021), 19–51. [Cited on pages 4, 69, and 74.]

[21] Borwein, J. M.; Moors, W. B.; and Wang, X., 1997. Lipschitz functions with

prescribed derivatives and subderivatives. *Nonlinear Analysis: Theory, Methods & Applications*, 29, 1 (1997), 53–63. [Cited on page 72.]

[22] Bottou, L.; Curtis, F. E.; and Nocedal, J., 2018. Optimization methods for large-scale machine learning. *Siam Review*, 60, 2 (2018), 223–311. [Cited on pages 12, 18, 20, 21, 24, and 92.]

[23] Boyd, S.; Boyd, S. P.; and Vandenberghe, L., 2004. *Convex optimization*. Cambridge university press. [Cited on pages 5, 9, and 15.]

[24] Bughin, J.; Seong, J.; Manyika, J.; Chui, M.; and Joshi, R., 2018. Notes from the ai frontier: Modeling the impact of ai on the world economy. https://www.mckinsey.com/featured-insights/artificial-intelligence/notes-from-the-AI-frontier-modeling-the-impact-of-ai-on-the-world-economy. [Cited on page 23.]

[25] Burke, J. V.; Curtis, F. E.; Lewis, A. S.; and Overton, M. L., 2019. The gradient sampling methodology. (2019). [Cited on pages 74 and 99.]

[26] Burke, J. V.; Lewis, A. S.; and Overton, M. L., 2002. Approximating subdifferentials by random sampling of gradients. *Mathematics of Operations Research*, 27, 3 (2002), 567–584. [Cited on pages 74, 81, and 99.]

[27] Carleo, G.; Cirac, I.; Cranmer, K.; Daudet, L.; Schuld, M.; Tishby, N.; Vogt-Maranto, L.; and Zdeborová, L., 2019. Machine learning and the physical sciences. *Reviews of Modern Physics*, 91, 4 (2019), 045002. [Cited on page 23.]

[28] Chartrand, R., 2011. Numerical differentiation of noisy, nonsmooth data. *International Scholarly Research Notices*, 2011 (2011). [Cited on pages 4 and 37.]

[29] Cheridito, P.; Jentzen, A.; and Rossmannek, F., 2021. Non-convergence of stochastic gradient descent in the training of deep neural networks. *Journal of Complexity*, 64 (2021), 101540. [Cited on pages 2 and 36.]

[30] Clarke, F. H., 1975. Generalized gradients and applications. *Transactions of the American Mathematical Society*, 205 (1975), 247–262. [Cited on pages 3, 4, 39, and 43.]

[31] Clarke, F. H.; Ledyaev, Y. S.; Stern, R. J.; and Wolenski, P. R., 2008. *Nonsmooth analysis and control theory*, vol. 178. Springer Science & Business Media. [Cited on pages 3 and 42.]

[32] Csáji, B. C. et al., 2001. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24, 48 (2001), 7. [Cited on page 30.]

[33] Cybenko, G., 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2, 4 (1989), 303–314. [Cited on page 30.]

[34] Dantzig, G. B. and Ramser, J. H., 1959. The truck dispatching problem. *Management science*, 6, 1 (1959), 80–91. [Cited on page 17.]

[35] Davis, D. and Drusvyatskiy, D., 2018. Stochastic subgradient method converges at the rate $o(k^{-1/4})$ on weakly convex functions. *arXiv preprint arXiv:1802.02988*, (2018). [Cited on pages 3, 92, and 94.]

[36] Davis, D.; Drusvyatskiy, D.; Kakade, S.; and Lee, J. D., 2020. Stochastic subgradient method converges on tame functions. *Foundations of computational mathematics*, 20, 1 (2020), 119–154. [Cited on pages 3, 48, 50, 51, 52, 54, 55, 56, 57, 58, 59, 60, 61, 63, 64, 65, 66, 98, and 101.]

[37] Douglas, J. and Rachford, H. H., 1956. On the numerical solution of heat conduction problems in two and three space variables. *Transactions of the American mathematical Society*, 82, 2 (1956), 421–439. [Cited on pages 3 and 44.]

[38] Duchi, J.; Hazan, E.; and Singer, Y., 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12, 7 (2011). [Cited on pages 3 and 44.]

[39] Duchi, J. C. and Ruan, F., 2017. Stochastic methods for composite optimization problems. *Preprint*, (2017). [Cited on pages 3 and 54.]

[40] El Ghaoui, L.; Gu, F.; Travacca, B.; Askari, A.; and Tsai, A., 2021. Implicit deep learning. *SIAM Journal on Mathematics of Data Science*, 3, 3 (2021), 930–958. [Cited on page 32.]

[41] Fermat, P., 1629. *Methodus ad disquirendam maximam et minimam*. Fermat, Samuel. [Cited on page 14.]

[42] Freitas, D.; Lopes, L. G.; and Morgado-Dias, F., 2020. Particle swarm optimisation: a historical review up to the current developments. *Entropy*, 22, 3 (2020), 362. [Cited on page 4.]

[43] Fürnkranz, J., 1997. Pruning algorithms for rule learning. *Machine learning*, 27, 2 (1997), 139–172. [Cited on page 26.]

[44] Georgii, H.-O., 2012. *Stochastics*. de Gruyter. [Cited on page 17.]

[45] Ghadimi, S. and Lan, G., 2013. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23, 4 (2013), 2341–2368. [Cited on pages 3, 4, 92, and 94.]

[46] Goldberg, Y., 2016. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57 (2016), 345–420. [Cited on page 25.]

[47] Gould, S.; Hartley, R.; and Campbell, D. J., 2021. Deep declarative networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (2021). [Cited on page 31.]

[48] Gratton, S.; Toint, P. L.; and Tröltzsch, A., 2011. How much gradient noise does a gradient-based linesearch method tolerate. Technical report, Citeseer. [Cited on page 37.]

[49] Harvey, N. J.; Liaw, C.; Plan, Y.; and Randhawa, S., 2019. Tight analyses for non-smooth stochastic gradient descent. In *Conference on Learning Theory*, 1579–1613. PMLR. [Cited on page 4.]

[50] Hendler, J., 2008. Avoiding another ai winter. *IEEE Intelligent Systems*, 23, 02 (2008), 2–4. [Cited on page 23.]

[51] Hinton, G. E. and Salakhutdinov, R. R., 2006. Reducing the dimensionality of data with neural networks. *science*, 313, 5786 (2006), 504–507. [Cited on page 24.]

[52] Hoerl, A. E. and Kennard, R. W., 1970. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12, 1 (1970), 55–67. [Cited on page 27.]

[53] Insights, F. B., 2021. Machine learning market size, share, growth, and trends [2029]. https://www.fortunebusinessinsights.com/machine-learning-market-102226. [Cited on page 1.]

[54] Jordan, M. I. and Mitchell, T. M., 2015. Machine learning: Trends, perspectives, and prospects. *Science*, 349, 6245 (2015), 255–260. [Cited on pages 23 and 24.]

[55] Kalantar, A.; Gelb, R. I.; and Alper, J. S., 1995. Biases in summary statistics of slopes and intercepts in linear regression with errors in both variables. *Talanta*, 42, 4 (1995), 597–603. [Cited on pages 93 and 94.]

[56] Karystinos, G. N. and Pados, D. A., 2000. On overfitting, generalization, and randomly expanded training sets. *IEEE Transactions on Neural Networks*, 11, 5 (2000), 1050–1057. [Cited on page 26.]

[57] Kingma, D. P. and Ba, J., 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, (2014). [Cited on pages 18, 81, and 98.]

[58] Kiwiel, K. C., 2007. Convergence of the gradient sampling algorithm for non-smooth nonconvex optimization. *SIAM Journal on Optimization*, 18, 2 (2007), 379–388. [Cited on pages 74 and 99.]

[59] Knowles, I. and Renka, R. J., 2014. Methods for numerical differentiation of noisy data. *Electronic Journal of Differential Equations*, 21 (2014), 235–246. [Cited on page 37.]

[60] Labrinidis, A. and Jagadish, H. V., 2012. Challenges and opportunities with big data. *Proceedings of the VLDB Endowment*, 5, 12 (2012), 2032–2033. [Cited on page 23.]

[61] Lacoste-Julien, S.; Schmidt, M.; and Bach, F., 2012. A simpler approach to obtaining an o (1/t) convergence rate for the projected stochastic subgradient method. *arXiv preprint arXiv:1212.2002*, (2012). [Cited on pages 3 and 80.]

[62] Lesh, F. H., 1959. Multi-dimensional least-squares polynomial curve fitting. *Communications of the ACM*, 2, 9 (1959), 29–30. [Cited on page 27.]

[63] Liu, R.; Gao, J.; Zhang, J.; Meng, D.; and Lin, Z., 2021. Investigating bi-level optimization for learning and vision from a unified perspective: A survey and beyond. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (2021). [Cited on page 32.]

[64] Liu, W.; Wang, Z.; Liu, X.; Zeng, N.; Liu, Y.; and Alsaadi, F. E., 2017. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234 (2017), 11–26. [Cited on page 29.]

[65] Maddison, C. J.; Mnih, A.; and Teh, Y. W., 2016. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, (2016). [Cited on page 10.]

[66] Mahoney, M. W.; Duchi, J. C.; and Gilbert, A. C., 2018. *The Mathematics of Data*, vol. 25. American Mathematical Soc. [Cited on page 24.]

[67] Mehrabi, N.; Morstatter, F.; Saxena, N.; Lerman, K.; and Galstyan, A., 2021. A survey on bias and fairness in machine learning. *ACM Computing Surveys (CSUR)*, 54, 6 (2021), 1–35. [Cited on page 25.]

[68] Moré, J. J. and Sorensen, D. C., 1982. Newton's method. Technical report, Argonne National Lab., IL (USA). [Cited on page 18.]

[69] Muller, M. E., 1959. A note on a method for generating points uniformly on n-dimensional spheres. *Communications of the ACM*, 2, 4 (1959), 19–20. [Cited on page 81.]

[70] Murtagh, F. and Contreras, P., 2012. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2, 1 (2012), 86–97. [Cited on page 33.]

[71] Nemirovskij, A. S. and Yudin, D. B., 1983. Problem complexity and method efficiency in optimization. (1983). [Cited on pages 3, 92, and 94.]

[72] Nesterov, Y. E., 1983. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. akad. nauk Sssr*, vol. 269, 543–547. [Cited on page 21.]

[73] Parker, R. G. and Rardin, R. L., 2014. *Discrete optimization*. Elsevier. [Cited on page 10.]

[74] Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito,

Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A., 2017. Automatic differentiation in pytorch. (2017). [Cited on page 4.]

[75] Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al., 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32 (2019). [Cited on pages 23, 36, and 75.]

[76] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E., 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12 (2011), 2825–2830. [Cited on page 23.]

[77] Prechelt, L., 1998. Early stopping-but when? In *Neural Networks: Tricks of the trade*, 55–69. Springer. [Cited on page 26.]

[78] Qian, N., 1999. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12, 1 (1999), 145–151. [Cited on pages 21, 81, and 98.]

[79] Qiu, J.; Wu, Q.; Ding, G.; Xu, Y.; and Feng, S., 2016. A survey of machine learning for big data processing. *EURASIP Journal on Advances in Signal Processing*, 2016, 1 (2016), 1–16. [Cited on page 23.]

[80] Qolomany, B.; Maabreh, M.; Al-Fuqaha, A.; Gupta, A.; and Benhaddou, D., 2017. Parameters optimization of deep learning models using particle swarm optimization. In *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, 1285–1290. IEEE. [Cited on page 4.]

[81] Rasamoelina, A. D.; Adjailia, F.; and Sinčák, P., 2020. A review of activation function for artificial neural network. In *2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*, 281–286. IEEE. [Cited on pages 30 and 83.]

[82] Rockafellar, R. T. and Wets, R. J.-B., 2009. *Variational analysis*, vol. 317. Springer Science & Business Media. [Cited on pages 3, 43, 46, and 98.]

[83] Ruder, S., 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, (2016). [Cited on page 18.]

[84] Ruiz, J. P. and Grossmann, I. E., 2017. Global optimization of non-convex generalized disjunctive programs: a review on reformulations and relaxation techniques. *Journal of Global Optimization*, 67, 1 (2017), 43–58. [Cited on page 10.]

[85] Ruszczyński, A., 2020. Convergence of a stochastic subgradient method with averaging for nonsmooth nonconvex constrained optimization. *Optimization Letters*, 14, 7 (2020), 1615–1625. [Cited on page 4.]

[86] Sammut, C. and Webb, G. I., 2011. *Encyclopedia of machine learning*. Springer Science & Business Media. [Cited on page 24.]

[87] Sard, A., 1942. The measure of the critical values of differentiable maps. *Bulletin of the American Mathematical Society*, 48, 12 (1942), 883–890. [Cited on page 58.]

[88] Schaller, R. R., 1997. Moore's law: past, present and future. *IEEE spectrum*, 34, 6 (1997), 52–59. [Cited on page 23.]

[89] Shamir, O. and Zhang, T., 2013. Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. In *International conference on machine learning*, 71–79. PMLR. [Cited on page 4.]

[90] Shi, Z.-J., 2004. Convergence of line search methods for unconstrained optimization. *Applied Mathematics and Computation*, 157, 2 (2004), 393–405. [Cited on page 21.]

[91] Shor, N. Z., 2012. *Minimization methods for non-differentiable functions*, vol. 3. Springer Science & Business Media. [Cited on page 73.]

[92] Smola, A. J. and Schölkopf, B., 2004. A tutorial on support vector regression. *Statistics and computing*, 14, 3 (2004), 199–222. [Cited on page 29.]

[93] Sturm, B. L.; Ben-Tal, O.; Monaghan, Ú.; Collins, N.; Herremans, D.; Chew, E.; Hadjeres, G.; Deruty, E.; and Pachet, F., 2019. Machine learning research that matters for music creation: A case study. *Journal of New Music Research*, 48, 1 (2019), 36–55. [Cited on page 23.]

[94] Sun, S.; Cao, Z.; Zhu, H.; and Zhao, J., 2019. A survey of optimization methods from a machine learning perspective. *IEEE transactions on cybernetics*, 50, 8 (2019), 3668–3681. [Cited on page 36.]

[95] Tibshirani, R., 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58, 1 (1996), 267–288. [Cited on pages 27 and 83.]

[96] Titchmarsh, E. C.; Heath-Brown, D. R.; Titchmarsh, E. C. T.; et al., 1986. *The theory of the Riemann zeta-function*. Oxford university press. [Cited on page 84.]

[97] Van den Dries, L. and Miller, C., 1996. Geometric categories and o-minimal structures. *Duke Mathematical Journal*, 84, 2 (1996), 497–540. [Cited on page 67.]

[98] Virtanen, P.; Gommers, R.; Oliphant, T. E.; Haberland, M.; Reddy, T.; Cournapeau, D.; Burovski, E.; Peterson, P.; Weckesser, W.; Bright, J.; et al., 2020. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17, 3 (2020), 261–272. [Cited on page 81.]

[99] Voelker, A. R.; Gosmann, J.; and Stewart, T. C., 2017. Efficiently sam-

pling vectors and coordinates from the n-sphere and n-ball. *Centre for Theoretical Neuroscience-Technical Report*, (2017). [Cited on page 81.]

[100] Wang, Q.; Ma, Y.; Zhao, K.; and Tian, Y., 2022. A comprehensive survey of loss functions in machine learning. *Annals of Data Science*, 9, 2 (2022), 187–212. [Cited on pages 76 and 83.]

[101] Wang, X., 2005. Subdifferentiability of real functions. *Real Analysis Exchange*, 30, 1 (2005), 137–171. [Cited on page 72.]

[102] Wilkie, A. J., 1996. Model completeness results for expansions of the ordered field of real numbers by restricted pfaffian functions and the exponential function. *Journal of the American Mathematical Society*, 9, 4 (1996), 1051–1094. [Cited on page 67.]

[103] Wolpert, D. H. and Macready, W. G., 1997. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1, 1 (1997), 67–82. [Cited on pages 8, 21, and 90.]

[104] Xia, L.; Anthonissen, M.; Hochstenbach, M.; and Koren, B., 2020. Improved stochastic rounding. *arXiv preprint arXiv:2006.00489*, (2020). [Cited on page 85.]

[105] Xu, Y.; Lin, Q.; and Yang, T., 2016. Accelerate stochastic subgradient method by leveraging local error bound. *CoRR, abs/1607.01027*, (2016). [Cited on pages 47 and 81.]

[106] Xu, Y.; Lin, Q.; and Yang, T., 2016. Accelerated stochastic subgradient methods under local error bound condition. *arXiv preprint arXiv:1607.01027*, (2016). [Cited on page 98.]

[107] Xu, Y.; Lin, Q.; and Yang, T., 2019. Accelerate stochastic subgradient method by leveraging local growth condition. *Analysis and Applications*, 17, 05 (2019), 773–818. [Cited on pages 47, 81, and 98.]

[108] Yadav, S. S. and Jadhav, S. M., 2019. Deep convolutional neural network based medical image classification for disease diagnosis. *Journal of Big Data*, 6, 1 (2019), 1–18. [Cited on page 23.]

[109] Ying, X., 2019. An overview of overfitting and its solutions. In *Journal of Physics: Conference Series*, vol. 1168, 022022. IOP Publishing. [Cited on page 26.]

[110] Young, S. R.; Rose, D. C.; Karnowski, T. P.; Lim, S.-H.; and Patton, R. M., 2015. Optimizing deep learning hyper-parameters through an evolutionary algorithm. In *Proceedings of the workshop on machine learning in high-performance computing environments*, 1–5. [Cited on page 23.]

[111] Zanakis, S. H. and Evans, J. R., 1981. Heuristic "optimization": Why, when, and how to use it. *Interfaces*, 11, 5 (1981), 84–91. [Cited on page 98.]

[112] Zaslavski, A. J. et al., 2016. *Numerical optimization with computational errors*, vol. 108. Springer. [Cited on page 85.]